НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО»

Факультет інформатики та обчислювальної техніки

Кафедра обчислювальної техніки

До захисту допущено:
Завідувач кафедри
Сергій СТІРЕНКО
«___»_____20__ р.

Дипломний проєкт

на здобуття ступеня бакалавра
за освітньо-професійною програмою «Комп'ютерні системи та мережі»
спеціальності 123 «Комп'ютерна інженерія»
на тему: «Програмно-апаратний комплекс для матричних операцій»

Виконала:
студентка IV курсу, групи ІО-64
Анастасія БРОВЧЕНКО _____

Керівник:
Доц. каф. ОТ, к. т. н.
Олександр КОРОЧКІН _____

Консультант з нормоконтролю:
Професор, доктор технічних наук
Валерій СІМОНЕНКО _____

Рецензент:
Доц. каф. СПКС, к. т. н., доц.
Оксана ТАРАСЕНКО-КЛЯТЧЕНКО _____

Засвідчую, що у цьому дипломному проекті немає запозичень з праць інших авторів без відповідних посилань.
Студентка _____

Київ – 2020 року

Національний технічний університет України
«Київський політехнічний інститут імені Ігоря Сікорського»
Факультет інформатики та обчислювальної техніки
Кафедра обчислювальної техніки

Рівень вищої освіти – перший (бакалаврський)

Спеціальність – 123 «Комп'ютерна інженерія»

Освітньо-професійна програма «Комп'ютерні системи та мережі»

ЗАТВЕРДЖУЮ
Завідувач кафедри
Сергій СТІРЕНКО
«___»_____2020 р.

ЗАВДАННЯ
на дипломний проєкт студентці
Анастасії БРОВЧЕНКО

1. Тема проекту «Програмно-апаратний комплекс для матричних операцій», керівник проєкту Корочкін Олександр Володимирович, доцент, к. т. н., затверджені наказом по університету від «07» травня 2020 р. №1081-с

2. Термін подання студентом проєкту 31.05.2020

3. Вихідні дані до проєкту технічна документація

4. Зміст пояснювальної записки

Аналіз апаратної частини високопродуктивних ПАК, аналіз програмної частини високопродуктивних ПАК. Розробка ПАК для матричних операцій. Тестування розробленого програмного забезпечення для ПАК.

5. Перелік графічного матеріалу

Схема топології ПАК, схема структурна процесора AMD 2950X, схема взаємодії потоків для задачі 1, схема взаємодії потоків для задачі 2, схема взаємодії потоків для задачі 3.

6. Консультанти розділів проекту

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
Нормоконтроль	Валерій СІМОНЕНКО		

7. Дата видачі завдання 01.09.2019

Календарний план

№ з/п	Назва етапів виконання дипломного проєкту	Термін виконання етапів проєкту	Примітка
1.	*Затвердження теми роботи*	01.09.2019	
2.	*Вивчення та аналіз завдання*	02.09.2019-13.04.2020	
3.	*Аналіз апаратного забезпечення ПАК*	13.04.2020-20.04.2020	
4.	*Аналіз програмного забезпечення ПАК*	21.04.2020-03.05.2020	
5.	*Реалізація системи*	04.05.2020-15.05.2020	
6.	*Оформлення пояснювальної записки*	13.04.2020-25.05.2020	
7.	*Передзахист*	26.05.2020	
8.	*Захист*		

Студент Анастасія БРОВЧЕНКО

Керівник Олександр КОРОЧКІН

Анотація

Бакалаврський дипломний проєкт присвячений розробці програмно-апаратного комплексу для виконання матричних операцій.

Комп'ютерна система складається з чотирьох ПК, об'єднаних в мережу з топологією «зірка з комутатором». Основою кожного ПК є 16-ядерний процесор AMD Ryzen Threadripper 2950X з підтримкою виконання двох потоків на кожному ядрі.

Розроблена програма для даної системи написана на мові C++ з використанням інтерфейсу MPI. Вона дозволяє максимально ефективно використовувати обчислювальну потужність системи, задіяти всі обчислювальні елементи і виконувати обчислення в рази швидше.

Annotation

Bachelor's project is devoted to the development of software-and-hardware system for matrix operations.

The computer system consists of four PCs connected in a network with a "star" topology. The heart of the "star" is switch. The basis of each PC is a 16-core AMD Ryzen Threadripper 2950X processor with hyper-threading technology – ability to perform two threads on each core.

The developed program for this system is written in C ++ using the MPI interface. It allows to make the most of the computing power of the system, to use all the computing elements and perform calculations much faster.

№ рядка	Формат	Позначення	Найменування	Кількість	Примітка
1	А4		Завдання на дипломний проект	2	
2	А4	ДП 4665.01.000.ВП	Відомість дипломного проекту	1	
3	А4	ДП 4665.02.000.ТЗ	Технічне завдання	3	
4	А4	ДП 4665.03.000.ПЗ	Пояснювальна записка	62	
5	А4	ДП 4665.04.000.Д1	Схема топології ПАК	1	
6	А4	ДП 4665.05.000.Д2	Схема структурна процесора AMD 2950X	1	
7	А4	ДП 4665.06.000.Д3	Схема взаємодії потоків в задачі 1	1	
8	А4	ДП 4665.07.000.Д4	Схема взаємодії потоків в задачі 2	1	
9	А4	ДП 4665.08.000.Д5	Схема взаємодії потоків в задачі 3	1	
10	А4	ДП 4665.09.000.Д6	Лістинг задачі 1	9	
11	А4	ДП 4665.10.000.Д7	Лістинг задачі 2	11	
12	А4	ДП 4665.11.000.Д8	Лістинг задачі 3	10	

Зм.	Арк.	№ докум.	Підпис	Дата	*ДП 4665.01.000.ВП*		
Розробила		Бровченко А.В.			*Програмно-апаратний комплекс для матричних операцій* *Відомість дипломного проекту*	Літ.	Аркуш 1 / Аркушів 1
Перевір.		Корочкін О.В.					
Н. контр.		Симоненко В.П.				НТУУ “КПІ”, ФІОТ, каф. ОТ, гр.ІО-64	
Затверд.		Корочкін О.В.					

Технічне завдання до дипломного проекту

ЗМІСТ

1. НАЙМЕНУВАННЯ ТА ОБЛАСТЬ ЗАСТОСУВАННЯ

Дане технічне завдання розповсюджується на розробку програмно-апаратного комплексу для матричних операцій.

Область застосування: розробка високопродуктивних комп'ютерних систем та створення для них спеціалізованого програмного забезпечення.

2. ПІДСТАВИ ДЛЯ РОЗРОБКИ

Підставою для розробки служить завдання на розробку програмно-апаратного комплексу для матричних операцій, затвердженою кафедрою обчислювальної техніки Національного технічного Університету України «Київський Політехнічний Інститут ім. Ігоря Сікорського».

3. МЕТА ТА ПРИЗНАЧЕННЯ РОЗРОБКИ

Метою даного проекту є розробка програмно-апаратного комплексу для матричних операцій.

4. ДЖЕРЕЛА РОЗРОБКИ

Джерелами для розробки служать науково-технічна література з комп'ютерних систем, публікації в періодичних виданнях, публікації в Інтернеті за даним питанням.

5. ТЕХНІЧНІ ВИМОГИ

5.1. Вимоги до розроблюваного продукту

- Швидкість виконання обчислень.
- Надійність системи, відмовостійкість.
- Паралельність – використання усіх обчислювальний елементів ПАК при виконанні програми.
- Можливість вдосконалювати систему.

5.2. Вимоги до програмного забезпечення

- Операційна система MS Windows XP, MS Windows Vista, MS Windows 7, MS Windows 8/8.1, MS Windows 10
- Інтерфейс MPI.

5.3. Вимоги до апаратного забезпечення

- Комп’ютер на базі процесору Intel Pentium 2 і вище
- Оперативної пам’яті не менше 128 Мбайт

Пояснювальна записка
до дипломного проєкту
на тему: «Програмно-апаратний комплекс для матричних операцій»

Київ – 2020 року

ЗМІСТ

ПЕРЕЛІК ТЕРМІНІВ ТА СКОРОЧЕНЬ

ПАК	Програмно-апаратний комплекс.
СЛАР	Система лінійних алгебраїчних рівнянь.
ПК (англ. PC)	(англ. Personal Computer) Персональний комп'ютер.
ЦП (англ. CPU)	(англ. Central Processing Unit) Центральний процесор.
ГП (англ. GPU)	(англ.Graphics Processing Unit) Графічний процесор.
ПКВМ (англ. FPGA)	(анггл. Field-Programmable Gate Array) Програмована користувачем вентильна матриця.
API	(англ. Application Programming Interface) набір визначень взаємодії різнотипного програмного забезпечення.

ВСТУП

Обчислення з використанням матриць та матричних операцій використовуються майже в усіх сферах нашого життя та науки. Їх використовують для моделювання молекулярної динаміки, поведінки комах у зграї, різноманітних космічних процесів, для прогнозування погоди чи розробки лікарських засобів.

Проблема матричних операцій в тому, вони є надто ресурсомісткими, потребують виконання великої кількості арифметичних операцій. Типовим прикладом є операція множення матриць. При деяких обчисленнях матриці бувають настільки великими, що один комп'ютер буде кілька днів, а то й довше, вираховувати потрібну задачу. Тому для подібних обчислень використовують суперкомп'ютери та обчислювальні кластери, що є значно потужнішими за звичайні ПК. Для них необхідно писати спеціальні багатопотокові програми, щоб використати їх обчислювальну потужність на повну.

З 2001 року набуло поширення використання графічних адаптерів для виконання арифметичних обчислень. Їх використання для матричних операцій є доволі перспективним з точки зору прискорення обчислень, оскільки в графічній карті є в десятки разів більше обчислювальних ядер, ніж в звичайному процесорі. Але ядра графічного процесору не можуть одночасно виконувати різні операції, наприклад додавання і множення. Одночасно може виконуватись лише одна операція над різними даними в кожному ядрі. Для матричних операцій таких спосіб обчислень не є недоліком, оскільки над кожним елементом матриці має виконатись одна й та ж арифметична операція.

Отже метою моєї роботи є розробка власної версії ПАК для обчислення матричних операцій, що дозволить скоротити час виконання цих операцій.

Для досягнення поставленої мети були поставлені наступні основні задачі:

- Провести аналіз сучасного апаратного та програмного забезпечення;
- Обрати апаратне забезпечення для виконання матричних операцій;
- Обрати програмне забезпечення для ПАК;
- Розробити програму за обраними параметрами;
- Проаналізувати розроблений ПАК для матричних операцій.

РОЗДІЛ 1. АНАЛІЗ АПАРАТНОГО ЗАБЕЗПЕЧЕННЯ ВИСОКОПРОДУКТИВНИХ ПАК

В цьому розділі розглянуто особливості різних комп'ютерних систем: багатоядерних, розподілених та з графічним адаптером. Розгляну недоліки і переваги усіх систем а також засоби для створення програмного забезпечення для кожної з них.

1.1 Багатоядерні системи

Багатоядерна система – це процесор, що має більше одного обчислювального ядра в своїй архітектурі. На сьогодні число ядер в процесорах доходить до 64. Існує 3 можливі варіанти багатоядерності:

- Незалежні процесорні ядра, кожне зі своєю кеш-пам'яттю, розташовані на одному кристалі і просто використовують спільну шину. Це 90-нм Pentium D на ядрі Smithfield.
- Кілька однакових ядер, розташованих на різних кристалах, що об'єднані разом в одному корпусі процесора (багаточіповий процесор). Таким є 65-нм покоління процесорів сімейств Pentium і Xeon на ядрах Presler і Dempsey.
- Ядра розташовані на одному кристалі і спільно використовують деякі ресурси кристала (шину, кеш-пам'ять). Такими є всі сучасні багатоядерні процесори.

Способи зв'язку між ядрами:

- Шина
- Мережа (Mesh) на каналах точка-точка
- Мережа з комутатором
- Спільна кеш-пам'ять

Спосіб зв'язку між ядрами впливає на швидкість передачі даних від одного ядра до іншого. Програмісту, що пише паралельну програму під конкретну систему, для створення ефективного алгоритму, слід добре вивчити особливості її архітектури.

На рис. 1.1 представлена масштабована архітектура Mesh (сітка), яка використовується в процесорах Intel з 2017 року [1]. До 2017 використовувалась кільцева архітектура.

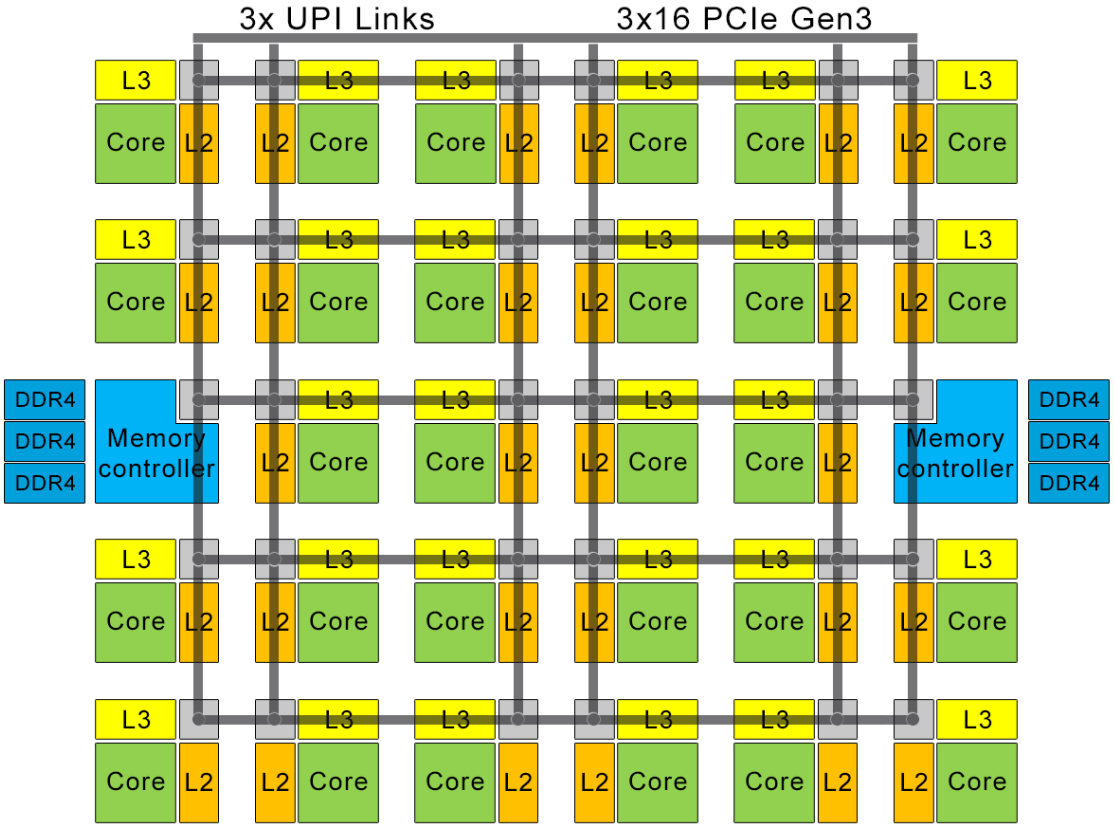


Рис. 1.1 – Архітектура Intel Mesh (сітка)

На рис 1.2 показано повнозв'язний комплекс з чотирьох ядер (CCX), що використовується в процесорах AMD з архітектурою Zen [2]. Пам'ять L3 є спільною для всіх ядер комплексу, але вона розділена на рівні частини для кожного ядра. Якщо ядру не вистачає своєї частини пам'яті воно може використовувати будь-яку L3 в рамках CCX. Згідно до архітектури Zen2 кожен кристал складається з двох таких комплексів, а ядро включає в себе один або кілька таких кристалів.

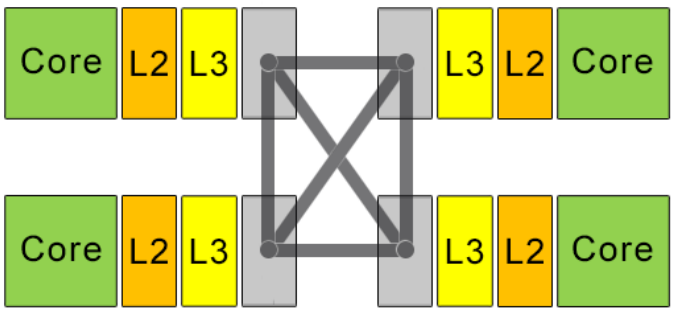


Рис. 1.2 - Core Complex архітектура AMD Zen 2

На сьогодні для багатьох виробників процесорів зокрема Intel, AMD, IBM, ARM збільшення числа ядер – є одним з пріоритетних напрямків збільшення продуктивності процесорів, так як інші напрямки гальмуються нинішнім рівнем розвитку науки і технологій.

В таблицях 1.1 - 1.4 наведено характеристики останніх поколінь багатоядерних процесорів для ПК від компаній Intel та AMD.

Таблиця 1.1 – Порівняння процесорів лінійки Intel Core X для ПК [3]

Номер процесора	**i9-10980XE**	**i9-9960X**	**i9-9980XE**	**i9-7960X**	**i9-7980XE**
Дата випуску	Q4'19	Q4'18	Q4'18	Q3'17	Q3'17
Кількість ядер	18	16	18	16	18
Кількість потоків	36	32	36	32	36
Тактова частота процесора	Базова 3.0 GHz Turbo Boost 4.6 GHz	Базова 3.1 GHz Turbo Boost 4.4 GHz	Базова 3.0 GHz Turbo Boost 4.4 GHz	Базова 2.8 GHz Turbo Boost 4.2 GHz	Базова 2.6 GHz Turbo Boost 4.2 GHz
Кеш-пам’ять	24.75 MB Intel® Smart Cache	22.00 MB Intel® Smart Cache	24.75 MB Intel® Smart Cache	22 MB	24.75 MB
Частота системної шини	8 GT/s	8 GT/s	8 GT/s	8 GT/s	8 GT/s
Ціна, грн (Rozetka)	31 550	43 763	60 640	43 999	99 999

Таблиця 1.2 – Порівняння процесорів лінійки Intel Core i9 [4]

Номер процесора	**i9-9900KS**	**i9-9900T**	**i9-9900**	**i9-9900K**
Дата випуску	Q4'19	Q2'19	Q2'19	Q4'18
Кількість ядер	8	8	8	8
Кількість потоків	16	16	16	16
Тактова частота процесора	Базова 4.00 GHz Turbo Boost 5.00 GHz	Базова 2.10 GHz Turbo Boost 4.40 GHz	Базова 3.10 GHz Turbo Boost 5.00 GHz	Базова 3.60 GHz Turbo Boost 5.00 GHz
Кеш-пам’ять	16 MB Intel® Smart Cache	16 MB Intel® Smart Cache	16 MB Intel® Smart Cache	16 MB Intel® Smart Cache
Частота системної шини	8 GT/s	8 GT/s	8 GT/s	8 GT/s
Ціна, грн (Rozetka)	20 893	-	13 750	18 831

Таблиця 1.3 – Порівняння процесорів лінійки AMD Threadripper 3rd Gen [5, 6]

Номер процесора	**3960X**	**3970X**	**3990X**
Дата випуску	Q4'19 (25, Nov)	Q4'19 (25, Nov)	Q1'20 (7, Feb)
Кількість ядер	24	32	64
Кількість потоків	48	64	128
Тактова частота процесора	Базова 3.80 GHz Макс. 4.50 GHz	Базова 3.70 GHz Макс. 4.50 GHz	Базова 2.90 GHz Макс. 4.30 GHz
Кеш-пам'ять	140 MB	144 MB	288 MB
Ціна, грн (Rozetka)	47 084	62 050	132 555

Таблиця 1.4 – Порівняння процесорів лінійки AMD Threadripper 2rd Gen [7, 8, 9]

Номер процесора	**2950X**	**2970WX**	**2990WX**
Дата випуску	Q3'18	Q3'18	Q3'18
Кількість ядер	16	24	32
Кількість потоків	32	48	64
Тактова частота процесора	Базова 3.50 GHz Макс. 4.40 GHz	Базова 3.00 GHz Макс. 4.20 GHz	Базова 3.00 GHz Макс. 4.20 GHz
Кеш-пам'ять	40 MB	80 MB	80 MB
Ціна, грн (Rozetka)	22 800	30 680	46 700

Щоб порівнювати процесори цих двох компаній, треба насамперед знати, що вони націлені на різні сегменти ринку процесорів для ПК: Intel працює над збільшенням продуктивності за рахунок збільшення тактової частоти процесорів, а AMD – над збільшенням кількості процесорних ядер і продуктивності за рахунок багатопотоковості.

Одними із головних показників продуктивності процесора є його тактова частота та частота системної шини. Частота системної шини впливає на швидкість передачі даних між процесорними ядрами (чим вища частота – тим вища швидкість передачі даних). Частота ядер процесора показує з якою швидкістю ядра виконують операції над даними. Проте процесор з більшою кількістю ядер і з меншою тактовою

частотою може бути продуктивнішим, ніж процесор з меншою кількістю ядер на більшою тактовою частотою.

Для виконання громіздких обчислень важливу роль грає об’єм кеш-пам’яті. Операції запису-читання до кеш-пам’яті процесора виконуються швидше, ніж ті ж операції у оперативну пам’ять.

1.2 Розподілені системи

Розподілена система – це набір незалежних комп'ютерів, що представляється їх користувачам єдиною об'єднаною системою, при цьому користувачам не відомі відмінності між комп'ютерами і способи зв'язку між ними. Системи зі спільною пам’яттю називаються мультипроцесорами (multiprocessors), а з розподіленою пам’яттю – мультикомп’ютерами (multicomputer). Типовий приклад мультикомп’ютера – кілька персональних комп'ютерів, об'єднаних в мережу. Типовий приклад розподіленої системи – інтернет. [10]

Характеристики топології мережі передачі даних:

- Кількість вузів/процесорів (P)
- Діаметр (D) – відстань між найвіддаленішими вузлами мережі. Ця величина може охарактеризувати максимально-необхідний час для передачі даних між вузлами/процесорами, оскільки час передачі зазвичай прямо пропорційний довжині шляху.
- Зв’язність (S) – кількість лінків, що йдуть до одного вузла.
- Вартість (R)– загальна кількість лінків (ліній передачі даних) в мережі.

Топології розподілених систем:

1. Лінія

Кількість процесорів $P = 4$

Зв’язність $S \leq 2$

Діаметр $D = P - 1 = 3$

Вартість $R = P - 1 = 3$

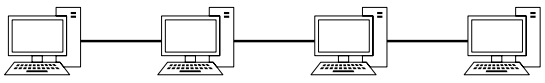


Рис. 1.3 – Лінійна топологія

Топологія лінія набула популярності завдяки своїй низькій вартості і легкості розширення мережі. Проте вона має один важливий недолік: фізичне пошкодження лише одного кабелю може вивести половину системи з ладу.

2. Кільце

Кількість процесорів $P = 4$

Зв'язність $S = 2$

Діаметр $D = \lfloor P/2 \rfloor = 2$

Вартість $R = P = 4$

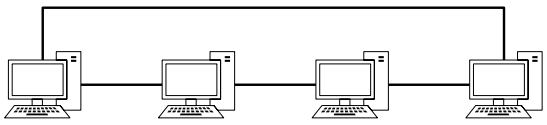


Рис. 1.4 – Топологія кільце

До переваг топології кільце можна теж віднести низьку вартість та легкість розширення, простоту установки. В такій системі шлях передачі даних між найвіддаленішими вузлами скорочується вдвічі. Серед недоліків: вразливість до фізичних пошкоджень кабелю.

3. Зірка

Кількість процесорів $P = 5$

Зв'язність $S \leq P - 1 = 4$

Діаметр $D = 2$

Вартість $R = P - 1 = 4$

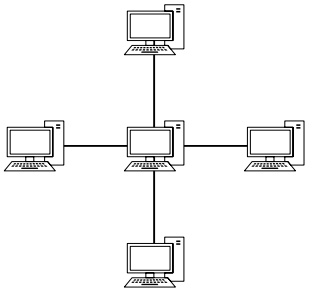


Рис. 1.5 – Топологія зірка

При виході з ладу одного кабелю, в такій системі з'єднання обірветься тільки одному користувачу, як наслідок – простий пошук несправностей і обривів. Серед переваг також простота підключення нових ПК. Щодо недоліків: несправність центрального вузла виведе з ладу всю систему; велика довжина сполучних кабелів; при великій кількості підключених ПК центральний вузол буде припадати високий трафік і він буде виконувати лише функцію передачі повідомлень між вузлами, а не обчислень, тому в такій системі буде доцільніше замінити центральний ПК на комутатор.

4. Решітка

Кількість процесорів $P = 9$

Зв'язність $S \leq 4$

Діаметр $D = 2\left(\sqrt{P} - 1\right) = 4$

Вартість $R = 2(P - \sqrt{P}) = 12$

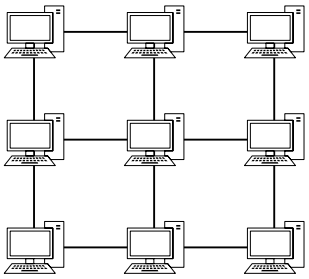


Рис. 1.6 – Топологія Решітка

Топологія решітка складається з n1 x n2 робочих станцій. Така система є доволі надійною, в порівнянні з попередніми. В ній легко шукати несправності. Використовується оптимальна кількість з'єднувальних кабелів. Серед недоліків: розширення системи можливе лише квантами розміру n1 чи n2.

5. Тор

Кількість процесорів $P = 9$

Зв'язність $S = 4$

Діаметр $D = 2 * \lfloor \sqrt{P}/2 \rfloor = 2$

Вартість $R = 2 * P = 18$

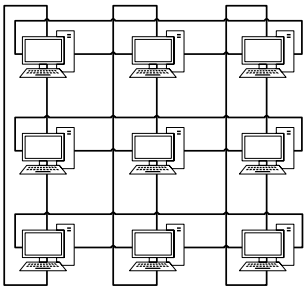


Рис. 1.7 – Топологія тор

Така топологія дуже схожа на решітку, за винятком додаткових лінків. Має ідентичні переваги та недоліки.

6. Повнозв'язна

Кількість процесорів $P = 4$

Зв'язність $S = P - 1 = 3$

Діаметр $D = 1$

Вартість $R = \frac{P(P-1)}{2} = 6$

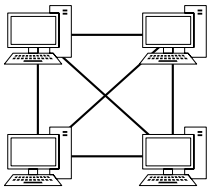


Рис. 1.8 – Повнозв'язна топологія

Єдиним плюсом такої системи є її діаметр: кожен вузол напряму з'єднаний з усіма іншими. Не зважаючи на свою логічну простоту, такий варіант з'єднання є громіздким та неефективним. Виникають труднощі при підключенні нового вузла. Найчастіше така топологія використовується для з'єднання невеликої кількості робочих станцій.

7. Гіперкуб

Кількість процесорів $P = 8$

Зв'язність $S = \log_2 P = 3$

Діаметр $D = \log_2 P = 3$

Вартість $R = \frac{P(\log_2 P)}{2} = 12$

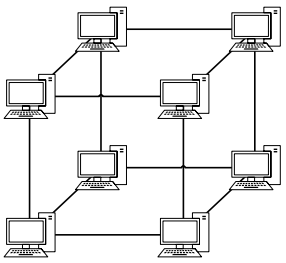


Рис. 1.9 – Топологія гіперкуб

Архітектура гіперкуб дає малу кількість лінків між процесорами. Така топологія є популярною при об'єднанні паралельних процесорів. Серед недоліків –

великі затрати при розширенні системи: з кожним порядком кількість вузлів і лінків подвоюється.

Основні характеристики розподілених систем:

- Спільне використання ресурсів – наприклад жорстких дисків, принтерів, файлів, компіляторів і т.п., зв'язаних за допомогою мережі.
- Відкритість – можливість розширювати систему шляхом додавання нових ресурсів (апаратного і програмного забезпечення від різних виробників).
- Паралельність – можливість одночасного виконання кількох процесів на різних комп'ютерах в мережі.
- Масштабованість – можливість нарощувати систему за допомогою додавання нових обчислювальних ресурсів.
- Відмовостійкість – стійкість до певних апаратних і програмних помилок. Більшість розподілених систем в разі помилки, як правило, можуть підтримувати хоча б часткову функціональність. Повний збій в роботі системи відбувається тільки в разі мережевих помилок.
- Прозорість – означає, що користувачам надано повністю прозорий доступ до ресурсів і в той же час від них прихована інформація про розподіл ресурсів в системі.

Здається, які можуть виникнути проблеми в таких системах – з’єднай кілька обчислювальних пристроїв, можливо комутатор, і запускай програми на виконання. Проте у розподілених систем є свої недоліки:

- Складність в проектуванні, побудові і відлагодженні їх роботи.
- Відмова компонентів системи. З часом деякі компоненти зношуються і відмовляють в роботі. Це призводить до перерозподілу задач на вузлах та зміни шляхів передачі даних, через що деякі ділянки такої системи можуть виявитися перевантаженими, а інші – простоювати.
- Зміни в топології. Розподілена система підлягає еволюції, наростає новими обчислювальними вузлами. Це призводить до тих же проблем, що і відмова вузла у відлагодженій системі.

1.3 Системи з графічним адаптером

Графічний процесор – окремий пристрій ПК, що призначений для виконання операцій з обробки графіки і обчислень з плаваючою крапкою.

В основі високої продуктивності графічних процесорів (GPU) лежить їх здатність виконувати програмний код паралельно на великій кількості однорідних обчислювальних елементів зі спільною пам'яттю. Подібна архітектура називається SIMT - Single Instruction Multiple Threads: код найчастіше однаковий, а оброблювані дані - різні. [11] Кожен обчислювальний елемент здатний виконувати тисячі потоків, перемикання між якими не має накладних витрат. Потоки можуть бути згруповані в блоки, які мають загальний кеш і швидку пам'ять, що розділяється і явно контролюється користувачем.

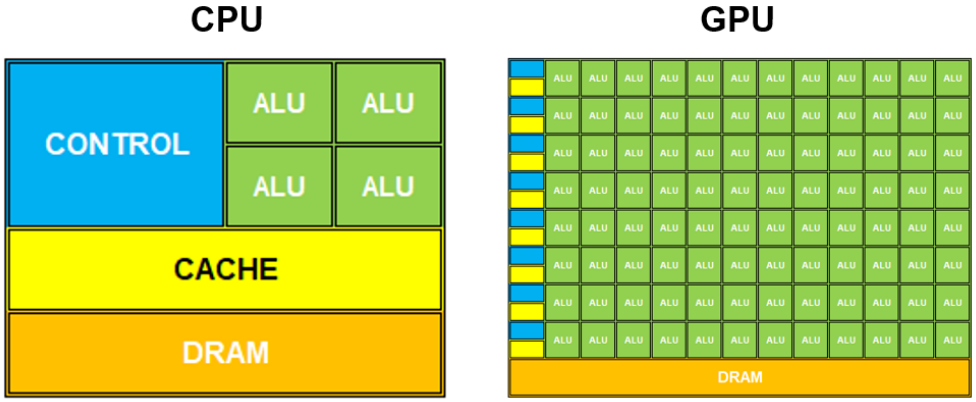


Рис. 1.10 – схематичне зображення чіпів CPU і GPU

В CPU велика частина площі чіпа зайнята під буфери команд, апаратне передбачення розгалуження і величезні обсяги кеш-пам'яті, а в GPU велика частина площі зайнята виконавчими блоками. Схематичне зображення чіпів CPU і GPU показано на рис. 1.10.

Основні характеристики відеокарт, що впливають на їх продуктивність:

- Пропускна здатність відеопам’яті.
- Тип відеопам’яті (GDDR4, GDDR5, GDDR6, HBM, HBM2 та інші) показує до якого покоління належить пам’ять GPU. Кожне наступне покоління є досконалішим за попереднє і забезпечує більш високу частоту роботи.
- Об’єм відеопам’яті. Якщо його недостатньо, то відеокарта починає використовувати оперативну пам’ять комп’ютера, доступ до якої займає значно

більше часу. Проте правило «чим більше – тим краще» в цьому випадку теж не працює. Необхідно щоб всі характеристики були збалансованими.

- Характеристики графічного ядра: тактова частота, кількість шейдерних процесорів. Тут вже діє правило «чим більше – тим краще».

На ринку з виробництва дискретних графічних процесорів фігурують всього дві компанії - NVIDIA і ATI / AMD. А їх партнери, такі як ASUS, Gigabyte, Sapphire, MSI та ін., займаються продажем відеокарт на базі GPU або від NVIDIA, або від ATI / AMD.

Розглянемо найкращі рішення відеокарт на момент написання цієї роботи [12]:

Таблиця 1.5 Порівняння відеокарт на основі GPU від AMD

Відеокарта	**ASUS ROG STRIX (RX5700XT-8G)**	**ASRock Phantom Gaming X Radeon VII**	**SAPPHIRE RADEON RX 5700 XT Nitro+**
GPU	RADEON RX 5700 XT	RADEON VII	RADEON RX 5700 XT
Частота GPU	Базова 1840 MHz GPU Boost 2035 MHz	Базова 1400 MHz GPU Boost 1750 MHz	Базова 1840 MHz GPU Boost 2035 MHz
Кількість шейдерних процесорів	2560	3840	2560
Пропускна здатність пам’яті	1024 Gb/s	1024 Gb/s	1024 Gb/s
Тип відеопам’яті	GDDR6	HBM2	GDDR6
Об’єм відеопам’яті	8 Gb	16 Gb	8 Gb
Розрядність шини відеопам’яті	256 bit	4096 bit	256 bit
Частота відеопам’яті	3500 MGz (14 GHz QDR)	1000 MHz	3500 MGz (14 GHz QDR)
Ціна, грн	11 111 (Rozetka)	22 449 (Brain)	14 328 (Rozetka)

Таблиця 1.6 Порівняння відеокарт на основі GPU від NVIDIA

Відеокарта	**MSI GAMING TRIO (RTX 2080 Ti GAMING TRIO)**	**GIGABYTE AORUS Waterforce Xtreme Edition (GV-N208TAORUS W-11GC)**	**ASUS ROG STRIX (ROG-STRIX-RTX2080TI-O11G-GAMING)**
GPU	GeForce RTX 2080 Ti	GeForce RTX 2080 Ti	GeForce RTX 2080 Ti
Частота GPU	Базова 1350 MHz GPU Boost 1635 MHz	Базова 1770 MHz	Базова 1350 MHz GPU Boost 1665 MHz
Кількість шейдерних процесорів	4352	4352	4352
Пропускна здатність пам’яті	616 Gb/s	616 Gb/s	616 Gb/s
Тип відеопам’яті	GDDR6	GDDR6	GDDR6
Об’єм відеопам’яті	11 Gb	11 Gb	11 Gb
Розрядність шини відеопам’яті	352 bits	352 bits	352 bits
Частота відеопам’яті	3500 MHz (14 GHz QDR)	3535 MHz (14.14 GHz QDR)	3500 MHz (14 GHz QDR)
Ціна, грн (Rozetka)	37 850	50 564	42 819

Графічні рішення від Nvidia швидше виконують обчислення, мають менше енергоспоживання і відповідно незначне виділення тепла. Карти від AMD роблять ставку на пропускну здатність пам'яті, але споживають більше енергії, виділяють більше тепла і не можуть змагатися за продуктивністю в high сегменті.

ВИСНОВКИ ДО РОЗДІЛУ 1

1. В даному розділі було проведено аналіз апаратного забезпечення для високопродуктивних ПАК.
2. Було розглянуто три типи систем: багатоядерна, розподілена, з графічним адаптером.
3. Проведений аналіз показав, що:
    a. Багатоядерна система є дорогим і малоефективним рішенням при об’ємних обчисленнях.
    b. Система з графічним адаптером є ефективною для вузького спектру задач. Наприклад для задачі сортування чи пошуку максимального/мінімального елементу така система є малоефективною.
    c. Розподілена система є відносно дешевим і ефективним рішенням, але розробка і відлагодження програм для такої системи є дуже об’ємним і складним процесом.

РОЗДІЛ 2. АНАЛІЗ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ ВИСОКОПРОДУКТИВНИХ ПАК

Для того, щоб використати увесь обчислювальний потенціал комп'ютерної системи, необхідно розробляти багатопотокові програми. Багато сучасних мов програмування підтримують багатопотоковість в тій чи іншій мірі. Для деяких мов необхідні додаткові знання для створення потоків та комунікації між ними. Інші ж намагаються повністю приховати процес розпаралелювання від розробника. Як не дивно, перші будуть ефективнішими за других.

У цьому розділі розглянто засоби створення програмного забезпечення для різних типів високопродуктивних ПАК.

2.1 Багатоядерні системи

Існують мови програмування, що підтримують багатопотоковість і мають власні засоби створення потоків. Мови високого рівня (Java, Python, .NET) надають потоковість розробнику у вигляді абстрактної специфічної платформи, що відрізняється від реалізації потоків в середовищі виконання. Ряд інших мов програмування (бібліотек) також намагається повністю абстрагувати концепцію паралелізму і потоковості від розробника (Cilk, OpenMP, MPI). Бібліотеки WinAPI, PThread, OpenMP надають розробнику інтерфейс для створення потоків.

2.1.1 Механізм моніторів в мові програмування Java

Для синхронізації потоків в Java використовується високорівневий механізм моніторів. Методи класів, що мають модифікатор synchronized виконуються в режимі взаємного виключення. Одночасно з одного класу може виконуватися лише один синхронізований метод. Інкапсуляція спільного ресурсу в такому класі-моніторі виконується за допомогою модифікатора private.

Ніякий інший потік не може увійти в блок, захищений монітором поки перший потік не вийде з synchronized-блоку. Якщо інші потоки викликають синхронізований метод, що вже виконується, вони будуть блоковані до завершення виконання цього синхронізованого методу.

```
public class Monitor {
   private Object obj = new Object();
   public void doSmth () {
       //... логіка, доступна для всіх потоків
       synchronized (obj) {
           //логіка, яка одночасно доступна лише для одного потоку
       }
   }
}
```

Якщо змінна є спільним ресурсом, оголошеним за допомогою модифікатора private або є volatile-змінною, то операції читання-запису такої змінної є атомарними.

Доступи до volatile-змінних впорядковані глобально, тобто кожен потік, який звертається до volatile-змінної, прочитає заново її значення, перед тим як продовжити виконання програми, замість того, щоб (по можливості) використати закешоване значення.

2.1.2 Засоби паралельного програмування в мові C#

Мова C# забезпечує програмування процесів через потоки. Програма мовою C# завжди є багатопотоковою, тому що виконання програми розпочинається з і створення нового потоку (“main” потоку) і породження додаткових потоків.

Для створення потоку в C# використовують клас Thread. Через конструктор класу передається метод, який буде виконуватись потоком (потоковий метод). Щоб запустити потік використовується метод Start(). Потік виконується незалежно і паралельно з іншими потоками. Метод join() дозволяє організувати очікування завершення викликаного потоку. Його зазвичай використовують в головному потоці, якщо потрібно, щоб він завершимся останнім.

Механізм семафорів в мові C# реалізований за допомогою класу Semaphore, який має 4 конструктори. Семафор створюється як об’єкт класу Semaphore:

```
Semaphore S = new Semaphore(p1, p2),
```

де p1 – початкове значення, p2 – максимальне значення семафору.

Для семафору визначені 21 метод. Методи S.WaitOne(), S.Resume() аналогічні операціям P(S), V(S).

Мютекс створюється як об’єкт класу Mutex (існує 5 конструкторів класу).

`Mutex M = new Mutex(false),` де false – початковий стан мютексу.

Для мютексів визначені 21 метод. Методи M.WaitOne(), M.ResumeMutex() аналогічні операціям P(M), V(M). [13]

Подія - мабуть, найпростіший і фундаментальний об'єкт для синхронізації потоків. Події з ручним скиданням реальзовані за допомогою класу ManualResetEvent.

```
ManualResetEvent E = new ManualResetEvent(false);
```

В конструктор передається початкове значення події. Методи E.WaitOne(), E.Set() аналогічні операціям P(E), V(E). Є також класи AutoResetEvent та EventWaitHandle. [14]

Механізм критичних секцій у C# реалізований за допомогою оператору `lock(Object O){ <блок коду> }`

В тілі описаний блок коду, виконання якого дозволяється лише одному процесу. Якщо процес виконує оператор lock, то він буде заблокований у випадку, коли інший процес вже розпочав виконання блоку коду з оператора lock з однаковим параметром Object O. Інша реалізація – за допомогою класу Monitor, у якому є методи Monitor.Enter(Object Mon), Monitor.Exit(Object Mon).[13]

2.1.3 Бібліотека WinAPI

Бібліотека Win32 (Windows API) входить до складу ОС Windows і містить набір функцій, які призначені для роботи з процесами.

Для створення процесу в Win32 використовують функцію CreateThread(). Функція повертає ідентифікатор процесу, який є унікальним і визначає його в системі. Під час створення процесу визначається початковий адрес коду, з якого має виконуватись потік. Зазвичай, це назва функції (потокової функції), яка вже створена і буде виконуватись як процес.

Функція:

```
HANDLE  Ім'я_Потоку = CreateThread(
     LPSECURITY_ATTRIBUTES атр, // атрибут безпеки
     SIZE_T     рс,                        // розмір стека
     LPTHREAD_STERT_ROUTUNE     фп,   // функція потоку
     LPVOID          афт,       // аргумент функції потоку
```

```
        DWORD      пр,        // прапорець виконання
        LPDWORD          іп);       // ідентифікатор потоку
```

Створює потік, для якого фактичні параметри визначають ім'я потокової функції та її параметри, атрибут безпеки, початковий розмір стека потоку, прапорець виконання: негайно (0) або відкладеного (Create_Suspended) , ім'я потоку.

Потік виконується доти, доки не відбудеться одна з таких подій:

- Функція повертає значення потоку;
- Потік викликає функцію ExitThread();
- Інший потік викликає функцію ExitProcess();
- Інший потік викликає функцію TerminateThread() з дескриптором потоку;
- Інший потік викликає функцію TerminateProcess() з дескриптором процесу.

Потік залишається в системі, поки він не закінчить роботу і всі його дескриптори не будуть закритими за допомогою функції CloseHandle(). [15]

В бібліотеці Win32 семафори є змінними спеціального типу HANDLE, за якими слідкує сама система. В цій бібліотеці лічильники семафорів також є багатозначними. Для семафорів визначені наступні операції.

`HANDLE CreateSemaphore (LPSECURITY_ATTRIBUTES lpSemaphore Attributes, LONG lInitialCount, LONG lMaximumCount, LPCTSTR lpName)` – операція, що створює семафор, де:

- lpSemaphoreAttributes – параметри захисту, зазвичай встановлено значення NULL.
- lInitialCount – початкове значення, воно має бути більше за 0 або рівним 0, та меншим за lMaximumCount або рівним йому. Стан семафору сигнальний, коли це число більше за 0 та не сигнальний, коли рівне 0. Значення семафору зменшується на 1 коли функція очікування розблоковує потік, який чекав на цей семафор. Значення семафору збільшується на визначене значення викликанням функції ReleaseSemaphore.
- lMaximumCount – максимальне значення, має бути більше за 0.
- lpName – ім'я семафору, яке має бути не більше за MAX_PATH та може містити в собі будь-які символи, окрім зворотного слешу(\).

`BOOL ReleaseSemaphore (HANDLE hSemaphore, LONG lReleaseCount, LPLONG lpPreviousCount)` – реалізація операції V(S).

- hSemaphore – семафор, який повертається функцією CreateSemaphore.
- lReleaseCount – число, на яке його потрібно збільшити, це число повинне бути більше за нуль. Якщо додавання цього числа до значення, яке вже є, зробить його більше за максимальне значення семафору, то додавання не відбудеться і функція поверне FALSE.
- lpPreviousCount – попереднє значення семафору. Може бути NULL, якщо попереднє значення не потребується. Якщо функція успішна, то значення, яке вона повертає, є ненульовим. Якщо ні, то повертає нуль.

Порівняння імен регістрозалежне. Якщо lpName співпадає з вже існуючим семафором, то lInitialCount та lMaximumCount ігноруються, тому що вони вже були встановлені при створенні. Якщо lpName є NULL, то семафор створюється без імені. Якщо функція успішна, то вона повертає семафор типу HANDLE. Якщо іменований об'єкт семафору існував до виклику функції, то функція GetLastError повертає ERROR_ALREADY_EXISTS. В інших випадках GetLastError повертає нуль. Якщо CreateSemaphore є невдалою, то повертає NULL. Для поширених відомостей про помилку потрібно викликати функцію GetLastError. [15]

`HANDLE CreateMutex( LPSECURITY_ATTRIBUTES lpMutexAttributes, BOOL bInitialOwner, LPCTSTR lpName)` – операція для створення мютексу, де:

- lpMutexAttributes – атрибут безпеки;
- bInitialOwner – початковий стан (прапор початкового власника);
- lpName – ім'я об'єкту.[16]

Подія - мабуть, найпростіший і фундаментальний об'єкт для синхронізації потоків. Це лише прапор, якому функціями SetEvent / ResetEvent можна задати стан: сигналізуючий або нейтральний.

`HANDLE CreateEvent (LPSECURITY_ATTRIBUTES lpEventAttributes, BOOL bManualReset, bInitialState, LPCTSTR lpName)` – операція для створення події, де

- lpEventAttributes – атрибут безпеки;
- bManualReset – тип скидання: TRUE – ручний;
- bInitialState – початковий стан: TRUE – сигнальний;
- lpName – ім'я об'єкту.

Якщо параметр bInitialState = FALSE, то буде створено подію з автоскиданням. Це означає, що як тільки якийсь потік, що очікує цієї події, буде звільнений сигналом від цієї події, вона автоматично буде скинута назад в нейтральний стан.[15]

Для роботи з критичною секцією потрібна змінна типу критичної секції (CRITICAL_SECTION).

До того, як почати використовувати критичну секцію, її треба проініціалізувати:
`VOID InitializeCriticalSection(LPCRITICAL_SECTION lpCriticalSection).`

Для оголошення початку критичної секції використовується функція:
`VOID EnterCriticalSection(LPCRITICAL_SECTION lpCriticalSection).`

Для виходу з критичної секції використовується функція:
`VOID EnterCriticalSection(LPCRITICAL_SECTION lpCriticalSection),`
де lpCriticalSection – вказівник на змінну типу CRITICAL_SECTION.

Існують класи-обгортки для критичних секцій: CLock, CAutoLock і CScopeLock. Класи CLock і CAutoLock зручно використовувати для синхронізації доступу до змінних класу, а CScopeLock призначений, в основному, для використання в процедурах. Зручно, що компілятор сам подбає про виклик :: LeaveCriticalSection() через деструктор.

`HANDLE WaitForSingleObject (HANDLE hHandle, DWORD dwMilliseconds)` – реалізація операції P(S), де:

- hHandle – ідентифікатор об’єкта;
- dwMilliseconds – визначає час тайм-ауту в мілісекундах. Функція повертає значення коли інтервал часу скінчується, навіть якщо стан семафору несигнальний. Якщо dwMilliseconds дорівнює нулю, то функція визначає стан семафору та повертається миттєво. Якщо dwMilliseconds дорівнює INFINITE, таймаут функції ніколи не наступає.

Якщо виконання функції успішне, то значення, що вона повертає, визначає яка подія змусила її повернутися. Є два такі значення: WAIT_OBJECT_0, яке повертається коли стан семафору стає сигнальним та WAIT_TIMEOUT, яке повертається коли трапляється тайм-аут та стан семафору є несигнальним. WAIT_FAILED вказує на не успішне виконання функції. Очікування на некоректний

семафор змушує функцію повернути WAIT_FAILED. Для поширених відомостей про помилку потрібно викликати функцію GetLastError.

`DWORD WaitForMultipleObjects( DWORD nCount, lpHandles CONST HANDLE *lpHandles, BOOL bWaitAll, DWORD dwMilliseconds)` - реалізація операції P(S), коли треба чекати на кілька об'єктів), де

- nCount - кількість об'єктів в масиві lpHandles;
- *lpHandles - вказівник на масив об'єктів;
- bWaitAll - прапорець, що вказує чи треба чекати на всі об'єкти чи можна дочекатись лише одного;
- dwMilliseconds - таймаут. [16]

2.2 Розподілені системи

Проблеми створення програмного забезпечення для розподілених систем:

- Затримки. Дані з різних вузлів приходять не одночасно і в різному порядку.
- Відмова компонентів системи при збереженні працездатності системи в цілому. Можливі випадки, коли один потік нескінченно довго чекає відповіді від іншого, нестабільного потоку.
- Виникнення «вузьких місць», коли на одну шину чи комутатор припадає значна частина пересилок даних.
- Зміни в топології. Розподілена система підлягає еволюції. Це може впливати на пропускну здатність, викликати проблеми з затримками.
- Гетерогенність. Архітектура вузлів системи може використовувати різні топології, механізми, програмне забезпечення.

2.2.1 Бібліотека MPI

Бібліотека (інтерфейс) MPI (Message Passing Interface) містить набір функцій, що дозволять організувати роботу з процесами в мовах, які не мають вбудованих засобів програмування процесів

Кількість процесорів, що використовуються, може бути задано вручну при запуску через командний рядок, а може визначатися в момент запуску програми засобами MPI-середовища.

Основні функції MPI, потрібні для створення процесів, наведено в табл. 1.7.

Таблиця 1.7 - Основні функції MPI

Функції	Дії
`MPI_Init()`	Підключити до MPI (параметри args і argv визначають аргументи програми). Завжди викликається першою.
`MPI_Comm_size()`	Визначити кількість процесів, які потрібно запустити
`MPI_Comm_rank()`	Отримати ранг (номер) процесу в групі (в комунікаторі)
`MPI_Finalize()`	Завершити виконання програми

Процес у MPI має назву задача. Всі задачі послідовно пронумеровані, починаючи з нуля. Номер задачі також називають рангом. Задачі в MPI можуть бути об'єднані в іменовану групу. Об'єкт - група дозволяє звертатися до групи як єдиного цілого (наприклад, відправляти повідомлення всім задачам групи), а також визначати дії, які виконуються тільки членами групи.

У рамках групи всі задачі мають унікальні ідентифікатори (ранги) - впорядковані числа, які розпочинаються з нуля. Спочатку всі задачі належать до однієї базової групи, з якої потім формуються нові групи- MPI надає набір функцій для роботи з групами.

Комунікатор - опис ділянки зв'язку процесів, які об'єднані в групу [13]. Програма може вміщувати кілька ділянок зв'язку. Нумерація процесів всередині ділянки зв'язку незалежна. Комунікатор може бути використаний як параметр MPI-функції для обмеження сфери її дії тільки заданою ділянкою зв'язку. Крім того, комунікатор забезпечує вимоги безпеки. MPI автоматично створює комунікатор MPI_COMM_WORLD, який є базовим для кожного додатка і створюється автоматично під час виклику функції MPI.Init() .

`int MPI_Comm_size(); int MPI_Comm_rank();` - для комунікатора повертають значення: size розміру групи (кількість задач, що приєднані до ділянки зв'язку) і rank - порядковий номер задачі, яка викликає цю функцію.

Одним з основних переваг в MPI є процес передачі та прийому повідомлень. Процеси в межах одного комунікатора можуть посилати та одержувати повідомлення. Передача повідомлень відбувається за допомогою функції:

```
MPI_Send(void* buf, int count, MPI_Datatype dtype, int destrank, int tag, MPI_Comm comm)
```

Функція виконує передачу count елементів типу dtype повідомлення з ідентифікатором tag процесу destrank в області зв'язку комунікатора comm.

Прийом повідомлень відбувається за допомогою функції:

```
int MPI_Recv(void* buf, int count, MPI_Datatype dtype, int sourcerank, int tag, MPI_Comm comm, MPI_Status *status)
```

Функція виконує отримання count елементів типу dtype повідомлення з ідентифікатором tag від процесу sourcerank в області зв'язку комунікатора comm.

2.2.2 Бібліотека PVM

Бібліотека PVM (Parallel Virtual Machine) є однією з перших систем програмування, що базується на механізмі передачі повідомлень і отримала широке поширення. Вона проектувалася для того, щоб зв'язати окремі незалежні комп'ютери у віртуальну обчислювальну систему, яка була б єдиним керованим обчислювальним ресурсом. Бібліотека PVM орієнтована на роботу з неоднорідними системами з розподіленою пам'яттю. В її основу покладена концепція віртуальної паралельної обчислювальної машини, яка об'єднує в єдиний ресурс безліч різнорідних вузлів. Цікаво, що віртуальна машина допускає динамічне конфігурування (причому користувачем, а не адміністратором). Є реалізації PVM для всіх основних операційних систем, і підтримується широкий набір мов програмування - Fortran, C / C ++, Tcl / Tk, Perl, Python. [17]

Система PVM складається з двох частин: PVM сервера (pvmd) і призначених для користувача бібліотек (libpvm3.a libfpvm3.a). Сервер pvmd забезпечує комунікації між комп'ютерами і управління процесами. На кожному комп'ютері віртуальної паралельної машини запускається один pvmd сервер. Перший pvmd сервер, що запускається користувачем, стає майстер-процесом, в той час як всі інші pvmd процеси, що запускаються майстром, називаються робітниками. PVM бібліотеки дозволяють процесам-робітникам взаємодіяти з pvmd серверами на інших вузлах. Вони містять функції для упакування і розпакування повідомлень і функції для передачі повідомлень.

Функція pvm_mytid() – повертає ідентифікатор процесу, з якого її викликано. `pvm_spawn(char *task, char **argv, int flag, char *where, int ntask, int *tids)` – запускає новий процес.

В PVM-середовищі для передачі даних між процесами використовуються буфери повідомлень. Кожен процес може мати один чи кілька таких буферів, але лише один з них є

активним. Перед відправкою кожного повідомлення викликається функція, яка дозволє підготувати чи сформувати активний буфер повідомлень:

```
pvm_initsend (int encoding)
```

Функції pvm_pk* використовуються для упаковки масиву заданого типу даних в активний буфер відправлення. Наприклад:

```
pvm_pkfloat( float *fp, int nitem, int stride)
```

Функції send і recv використовуються для відправки і отримання повідомлень відповідно.

```
pvm_send(int tid, int msgtag)
pvm_recv(int tid, int msgtag)
```

Функції pvm_upk* виймають отримане повідомлення з активного буферу повідомлень і розпаковують його, зберігаючи в масиві вказаного типу. Наприклад:

```
pvm_upkfloat( float *fp, int nitem, int stride)
```

При завершенні виконання процесу викликається pvm_exit(). Це дозволяє не просто завершии процес, а й від'єднати його від PVM-середовища.

2.2.3 Механізм Randevu в мові програмування ADA

Механізм рандеву визначає реалізацію взаємодії задач у мові Ада через посилання повідомлень. Оператори входу entry, прийняття виклику входу accept, оператор відбору select дозволяють ефективно реалізувати різноманітні можливості моделі рандеву.

Необхідно звернути увагу на несиметричність такого механізму взаємодії. Це означає, що в процесі взаємодії одна з задач розглядається як сервер, а інша - як клієнт, причому задача-сервер не може бути ініціатором початку взаємодії.

Оскільки задача-клієнт і задача-сервер виконуються незалежно один від одного, то немає ніякої гарантії, що обидві задачі виявляться в точці здійснення рандеву одночасно. Тому, якщо задача-сервер опинилась в точці рандеву, але при цьому немає жодного звернення до входу (запиту на взаємодію), то вона повинна чекати появи такого звернення. Аналогічно, якщо задача-клієнт звертається до входу, а задача-сервер не готова обслужити таке звернення, то задача-клієнт повинна чекати, поки задача-сервер обслужить це звернення. В процесі очікування, як задача-клієнт, так і

задача-сервер не займають ресурси процесора, перебуваючи в стані, який називають станом блокування. [18]

Приклад використання механізму рандеву для організації взаємодії задач T1 і T2, де задача T1 передає в задачу T2 ціле число x. Для взаємодії задач у специфікації задачі T2 описаний вхід DataIn, специфікація якого дозволяє приймати дані в задачі B. Посилання цілого числа із задачі A виконано як виклик входу DataIn з підстановкою фактичного параметру.

Структурну схему взаємодії задач, під час якої здійснюють обмін даними зображено на рис. 2.1. При цьому застосовані два входи: DataIn для передавання даних і ResultOut для повернення результату. Для запуску задач їх вкладено в процедуру Run_Tasks, виклик якої приведе до старту задач T1 і T2.

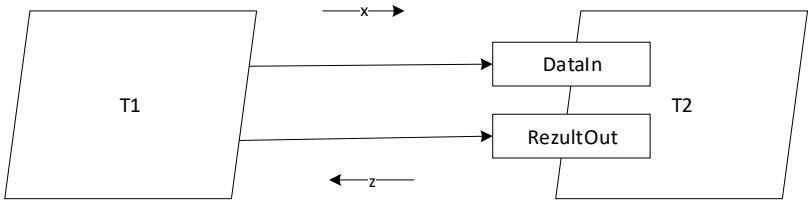


Рис. 2.1 Схема взаємодії задач (двостороння)

Структурну схему взаємодії трьох задач T1, T2 і T3 зображено на рис. 2.2. Задача T3 приймає данні від задач T1 і T2 за допомогою входів: DataInT1 і DataInT2. Особливістю реалізації взаємодії задач є використання в тілі задачі T3 оператору select. Це дозволяє задачі T3 приймати виклик входів DataInT1 і DataInT2 в будь якій послідовності в міру їх надходження. [18]

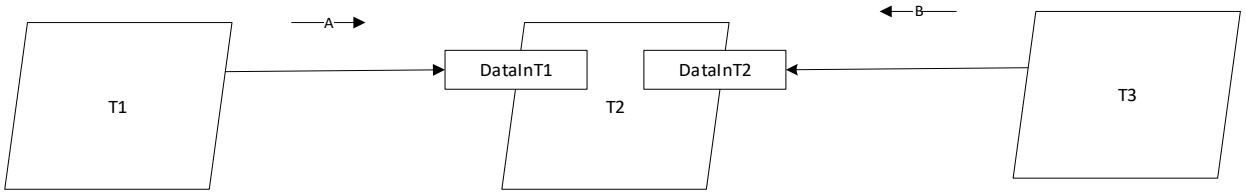


Рис. 2.2 Схема взаємодії задач

2.3 Системи з графічним адаптером

Спосіб використання графічного процесора для виконання арифметичних операцій потребував представлення вхідних даних у графічному вигляді і перетворення вихідних графічних даних в числові. Такі перетворення підтримуються

двома програмними інтерфейсами для графічних процесорів: OpenGL і DirectX. Ці громіздкі перетворення були усунені з появою мов програмування загального призначення та таких інтерфейсів як: Sh/RapidMind, Brook та Accelerator. Сучасні GPU можуть виконувати будь-які операції з великими даними і повноцінно використовувати свою швидкість, не вимагаючи представлення даних в графічній формі.

Сучасні програмні інтерфейси для GPGPU:

- OpenCL – відкритий стандарт для розробки програм як на CPU, так і на GPU, розроблений на мові програмування C. Апаратно і програмно-незалежна платформа.
- DirectCompute – прикладна мова програмування, що є частиною DirectX і підтримується починаючи з 10 версії DirectX.
- C++ AMP – бібліотека, розроблена компанією Microsoft на мові C++, для роботи необхідний DirectX 11.
- CUDA – фреймворк для програмно-апаратних обчислень від компанії NVIDIA на мові C. Апаратно-залежна платформа, тільки для відеокарт від компанії NVIDIA.
- AMD FireStream – фреймворк для програмно-апаратних обчислень від компанії AMD. Апаратно-незалежна платформа.

2.3.1 CUDA

Програми в архітектурі CUDA пишуться «розширеною» мовою C, а для їх компіляції використовується спеціальний компілятор nvcc. Розширення мови включає специфікатор функцій (вони показують, де буде виконуватися функція, і звідки може бути викликана), додаткові типи даних, специфікатор змінних (вони визначають, яка пам'ять буде використовуватися для їх розміщення), синтаксис запуску ядер, вбудовані змінні, що зберігають інформацію про кожну нитку, а також математичні функції.

До речі, незважаючи на те, що за поняттям thread в нашій літературі закріпився переклад потік, тут (і в перекладній літературі по CUDA) застосовується переклад нитка, оскільки в рамках CUDA поняття потік (stream) також використовується.

Функції, що виконуються багатьма потоками-нитками, будуть позначені на програмі специфікаторами __global__ і можуть бути викликані на виконання за допомогою спеціального нового синтаксису:

```
<< . . . >>:
// визначення функції-ядра
__global__ void MyKernel( . . . )
{
. . .
}
. . .
// виклик ядра на виконання
MyKernel<<M, N>>(...);
. . .
```

Виклик ядер на виконання ініціюється в програмі згадкою їх імені з так званими параметрами конфігурації запуску ядер (які вказуються в потрійних кутових дужках <<< ... >>>) поряд зі звичайними параметрами, необхідними для роботи кожного окремого ядра. Для позначення того, що функція є ядром (тобто, повинна виконуватися на GPU і може бути запущена на виконання з CPU), використовується специфікатор __global__ . Специфікатори __host__ і __device__ позначають функції, виконувані на CPU і GPU відповідно, причому і викликані вони можуть бути тільки звідти, де виконуються.

Для завдання розміщення в пам'яті GPU змінних використовуються специфікатори __device__ (змінна знаходиться в глобальній пам'яті і доступна всім ниткам), __constant__ (розміщена в так званій константної пам'яті, звідки вона може бути тільки прочитана будь-якою з ниток), __shared__ (змінна - в розподіленій пам'яті GPU, де доступна лише всім ниткам «свого» блоку).

Додаткові типи даних CUDA - це 1/2/3/4-вимірні вектори з базових типів мови C (char, short, int, long - unsigned і signed, longlong, float і double), імена їх утворюються додаванням цифри з числом компонент: char1, char2, char3, char4.

Звернення до компонентів здійснюється за іменами (x, y, z, w), а створення значень векторів - за допомогою виклику функцій make_<TypeName>, де <TypeName> - один з вищенаведених типів даних, наприклад:

```
int2 a = make_int2(1,4);
float3 u = make_float3(1,2.72f,3.14f);
```

Для задання розмірності є тип dim3, він має також нормальний конструктор:

```
dim3 Blocks(16,16); // <=> Blocks(16,16);
dim3 Grid(256); // <=> Grid(256,1,1);
```

Повний синтаксис запуску ядер на виконання виглядає так:

```
kernelName<<Dg, Db, Ns, S>>(args);
```

Тут kernelName - ім'я __global__- функції, Dg - змінна типу dim3 (параметри сітки блоків), Db - змінна типу dim3 (параметри блоку ниток), Ns - додатковий обсяг розподіленої пам'яті в байтах для кожного блоку (необов'язковий параметр, він за замовчуванням дорівнює нулю), S - задає потік (CUDAstream), в якому повинен відбутися виклик (потік 0 за замовчуванням), args - параметри функції-ядра (їх може бути декілька, але загальний розмір цих параметрів поки обмежений 256 байтами).

Способи виділення розподіленої пам’яті в ядрі:

- Явно вказати розмір масиву зі специфікатором __shared__.
- При запуску ядра задати додатковий обсяг розподіленої пам'яті в байтах, який необхідно виділити кожному блоку (третій параметр конфігурації функції-ядра). Для доступу в ядрах до такої пам'яті використовується опис масиву в функції ядра без явно заданого розміру, наприклад: `__shared__ float buf[];`

При цьому сам виклик ядра (з назвою kernel) виглядає так:

```
kernel<<< ... , ... , k*sizeof(float) >>>( ... );
```

Функція __syncthreads() – точка синхронізації - працює як бар’єр, перед яким всі нитки блоку мають запунитись, в очікуванні завершення роботи інших ниток блоку.

Кожна копія функції ядра після запуску на виконання має доступ до спеціальних змінних: до розмерностей блоків і сітки (вони називаються blockDim і gridDim відповідно, обидві мають тип dim3), а також до положень конкретної копії ядра в блоці і цього блоку в усій сітці при виконанні коду (threadIdx, blockIdx, обидві змінні мають тип int3). Використовуючи ці змінні, кожна копія ядра може сформувати унікальні індекси для доступу до даних. [11]

2.3.2 OpenCL

OpenCL (Open Computing Language) - це відкритий стандарт для паралельного програмування і роботи з широким набором сучасних паралельних обчислювачів (багатоядерних процесорів, GPU, FPGA). Спочатку OpenCL був запропонований фірмою Apple, але згодом отримав підтримку багатьох представників галузі, в тому числі Intel, AMD, IBM, NVIDIA, ARM, Samsung і ін.

Робота програми, що використовує OpenCL, протікає приблизно так: опитуються наявні обчислювальні ресурси, обираються ті, що будуть далі використовуватися, з вихідного коду створюються обчислювальні ядра і розподіляються для запуску на обчислювальні ресурси.

Таким чином, розробка OpenCL-програми зводиться до написання ядер і host-додатку для ПК, який розподіляє потрібні ядра на доступні пристрої. Такий додаток повинен використовувати п'ять структур: cl_device_id, cl_kernel, cl_program, cl_command_queue, cl_context. Він розподіляє ядра (cl_kernel), отримані з вихідного коду (cl_program) на прилади (cl_device_id), ці ядра потрапляють на пристрої через чергу команд (cl_command_queue); контекст (cl_context) дозволяє пристроям отримувати ядра і обмінюватися даними.

Подібний додаток має отримати дані про пристрій, що буде далі виконувати функцію-ядро, наприклад, як описано нижче.

Перша виявлена платформа:

```
clGetPlatformIDs(1, &platform, NULL);
```

Перший пристрій (GPU):

```
clGetDeviceIDs(platform, CL_DEVICE_TYPE_GPU, 1, &device, NULL);
```

Далі додаток створює контекст, наприклад, тільки з одним виявленим пристроєм:

```
context = clCreateContext(NULL, 1, &device, NULL, NULL, &err);
```

`&err` – адреса змінної, в яку запишеться код помилки.

Після цього додаток повинен отримати програму з коду ядра, що міститься, наприклад, у файлі hello.cl, для чого вміст цього файлу читається в масив, який передається функції:

```
clCreateProgramWithSource:
```

```
program = clCreateProgramWithSource(context, 1, (const char**)&program_buffer, &program_size, &err);
clBuildProgram(program, 0, &device, NULL, NULL, NULL);
```

Відсутні в останньому виклику параметри можуть визначати варіанти компіляції. Після неї з заданої функції створюється ядро (тут "hello" - назва функції ядра):

```
kernel = clCreateKernel(program, "hello", &err);
```

Для розподілу ядер на прилади необхідно створювати черги:

```
queue = clCreateCommandQueue(context, device, 0, &err);
```

Оскільки ядру знадобиться пам'ять для виведення, необхідно також створити буфер пам'яті:

```
mem = clCreateBuffer(context, CL_MEM_READ_WRITE, MEM_SIZE * sizeof(char), NULL, &ret);
```

Коли всі компоненти оточення (тобто, структури cl_device_id, cl_kernel, cl_program, cl_command_queue, cl_context) створені, слід підготувати ядру необхідні параметри виклику, наприклад, в разі ядра hello - це один параметр (адреса буфера пам'яті для виведення):

```
ret = clSetKernelArg(kernel, 0, sizeof(cl_mem), (void *)&mem);
```

і можна відправляти ядро в чергу на виконання:

```
global_size = 8;
local_size = 4;
clEnqueueNDRangeKernel(queue, kernel, 1, NULL, & global_size, &local_size, 0, NULL, NULL);
```

Остання функція не тільки забезпечує запуск ядра на пристрої, а й визначає, як багато робочих одиниць має бути створено (параметр global_size), а також скільки робочих одиниць буде в робочій групі (параметр local_size). Для читання отриманих результатів з буфера пам'яті mem в масив (в даному випадку - символів) string викликається функція clEnqueueReadBuffer:

```
ret = clEnqueueReadBuffer(queue, mem, CL_TRUE, 0, MEM_SIZE * sizeof(char), string, 0, NULL, NULL);
```

Вона повертає значення коду можливої помилки (або значення CL_SUCCESS в разі успішного завершення). [11]

ВИСНОВКИ ДО РОЗДІЛУ 2

1. В даному розділі було проведено аналіз програмного забезпечення для високопродуктивних ПАК.
2. Було розглянуто ПЗ для трьох типів систем: багатоядерної, розподіленої, з графічним адаптером.
3. Проведений аналіз показав, що:
    a. Процес розробки програм для багатоядерної система є найпростішим серед розглянутих. Деякі мови програмування навіть мають вбудовані засоби паралельного програмування, а для інших можна використовувати сторонні бібліотеки.
    b. Використання систем з графічним адаптером для виконання арифметичних обчислень стало популярним не так давно – після 2001 року і зараз продовжує поширюватись. Фреймворк OpenCL є універсальним і тому на сьогодні найпопулярніший для GPGPU обчислень.
    c. Не зважаючи на те, що розробка і відлагодження програм для розподіленої системи є складним процесом, існує немало ефективних рішень для спрощення їх реалізації. Найпопулярніші з них: MPI, PVM.

РОЗДІЛ 3. ПРОЕКТУВАННЯ ТА РОЗРОБКА ПАК.

В цьому розділі проводиться вибір апаратного та програмного забезпечення для ПАК для виконання матричних операцій.

В математиці матриці використовуються для компактного запису СЛАР та систем диференціальних рівнянь. СЛАР можна вирішити за допомогою методу Гауса, що базується на матричних операціях, чи безпосередньо звести рішення СЛАР до матричних операцій. В економіці їх використовують для створення баз даних та для економічних досліджень, коли головним об'єктом дослідження є балансове співвідношення затрат і результатів діяльності. В статистиці та в машинному навчанні матриці використовуються при обчисленні методу множинної лінійної регресії, коли на одну змінну впливає кілька незалежних між собою факторів. В біології, фізиці, хімії, астрономії матриці використовуються для моделювання різноманітних процесів: молекулярної динаміки, моделювання поведінки комах у зграї, прогнозуванні погоди, моделювання космічних процесів. Використовують матриці для кодування інформації – пам'ять з кодом усунення помилок (ECC). При програмуванні графи представляють у вигляді матриць переходів від однієї вершини до іншої. В теорії імовірності матриці використовують для представлення імовірності переходів в дискретних ланцюгах Маркова. В теорії управління використовуються матриці для визначення траєкторії руху об'єкта в просторі. Будь-яка картинка на екрані – це матриця, елементами якої є кольори на екрані. Також за допомогою матричних операцій можна знайти новий об'єкт на космічному знімку чи відфільтрувати цей знімок від шумів. Освітлення, падаючі тіні на героїв комп'ютерної гри – це також результат матричних операцій.

Комп'ютерна система для матричних операцій має відповідати таким вимогам:

- Простота реалізації;
- Висока продуктивність;
- Оптимальність співвідношення ціна-якість;
- Зручність обслуговування.

3.1 Вибір апаратного забезпечення

3.1.1 Тип системи

В першому розділі було розглянуто типи комп'ютерних систем. Далі детальніше розглянемо переваги і недоліки кожної.

Багатоядерна система зручна в зборці: потрібно зібрати один набір комплектуючих для ПК. Для такої системи необхідно обрати потужний процесор, що, звичайно, недешеве задоволення. При розробці ПЗ для такої системи буде простішою, ніж для інших розглянутих типів комп'ютерних систем. Але її продуктивність буде обмеженою продуктивністю лише одного процесора.

Для обчислювального кластера (розподіленої системи) необхідно зібрати кілька ПК і обрати обладнання для з'єднання цих ПК. Основними перевагами кластерних обчислень є дешевизна та легке розширення системи. Продуктивність цієї системи можна нарощувати з часом, підключаючи нові вузли. Мінусом є те, що при розробці ПЗ необхідно враховувати еволюційні зміни в топології: деякі вузли можуть відмовити з часом (не виключаємо виробничий брак), з'являться нові вузли, можливо зміниться сама топологія.

Для системи з графічним адаптером треба обрати хорошу відеокарту. Виконання обчислень на відеокарті не означає, що CPU не буде брати участь в обчисленнях, тому CPU треба обирати відповідно з гарною продуктивністю. Такій системі необхідних потужніший блок живлення. Для написання ефективних паралельних програм і їх відлагодження необхідні додаткові знання і вміння. Собівартість такої системи буде однозначно дорожчою за багатоядерну.

Отже, зважаючи на всі описані переваги і недоліки, було вирішено збирати обчислювальний кластер на основі мережі кількох ПК. Така система є відносно недорогою, простою в реалізації та подальшій модернізації. Її можна легко зібрати за необхідності. А щодо програмного забезпечення, наразі існують інструменти, API, що полегшують розробку програм для розподілених комп'ютерних систем.

Всі ПК будуть об'єднані в мережу з топологією зірка, де роль центрального вузла буде виконувати комутатор (додаток А). Перевагою такої топології є те, що передачею повідомлень буде займатись виключно комутатор, обчислювальні вузли

не будуть переривати обчислення для передачі повідомлення від одного сусіда іншому.

3.1.2 Апаратне забезпечення

Систему будемо будувати на 16-ядерному процесорі AMD Ryzen Threadripper 2rd Gen 2950X. Процесор має 4 канали пам'яті, тому обираємо 4 планки ОЗУ по 8Гб. Так як відсутня вбудована графіка, необхідно обрати дискретну відеокарту. Для наших потреб підійде найпростіша та найдешевша на Розетці на даний момент GeForce GT 710. До материнської плати особливих вимог немає, обираємо по необхідному сокету Gigabyte X399 Aorus Pro. В якості жорсткого диску обираємо SSD з інтерфейсом M.2, що підтримується материнською платою, так як швидкість передачі даних в M.2 є більшою, ніж у звичному SATA. Процесори від AMD відомі своїм значним тепловиділенням під час роботи, тому систему охолодження необхідно обрати відповідно хорошу. Рідинне охолодження є ефективнішим за повітряне, тому обираємо одну з рекомендованих компанією AMD рідинних систем для цього процесора: ID-COOLING Zoomflow 240. Блок живлення підбираємо необхідної потужності.

Розрахуємо собівартість одного ПК в такій системі (табл. 2.1) і всієї системи вцілому. Можна обрати будь-яку кількість ПК для об'єднання в систему. В якості прикладу було обрано 4 ПК.

Таблиця 2.1 – Розрахунок собівартості ПК

	Ціна
Материнська плата Gigabyte X399 Aorus Pro	9 550
Процесор Threadripper 2950X	22 800
Графічний адаптер GeForce GT 710 1GB	1 069
Оперативна пам'ять 4x8Гб	5 500
SSD Kingston SSD SSDNow A400 120GB M.2	839
Блок живлення Cooler Master MWE 500	1 616
Система охолодження ID-COOLING Zoomflow 240	2 060
Всього :	**43 434**

Отже вартість одного ПК складає 43 434 грн.

Вартість системи = 4 * 43 434 = 173 736 грн.

3.1.3 Особливості архітектури процесора

Усі процесори AMD Ryzen Threadripper другого покоління мають архітектуру під кодовою назвою Zen+ і 12 нм. техпроцес. Ця архітектура прийшла на зміну архітектурі Zen. Мікропроцесори Zen+ працюють на вищих тактових частотах і мають при цьому нижче енергоспоживання. Конвеєр Zen+ залишився ідентичним до Zen.

Основою процесорів з такою архітектурою є кристал Zeppelin. Кілька таких кристалів комутуються за допомогою шини Infinity Fabric, що працює з дійсною частотою ОЗУ. Основою кристала Zeppelin є 2 блоки Core Complex (CCX) і загальний кеш 3го рівня (L3). У кожному CCX розташовані 4 ядра Zen із загальним для всіх ядер кешем третього рівня, об'ємом 8 МБ на комплекс. Кожне ядро в комплексі може звернутися до кеша будь-якого рівня приблизно з однаковою швидкістю, проте в рамках CCX є деяке уповільнення при зверненні до дальньої 4МБ половини L3 кешу, а доступ до 8 МБ L3 пам'яті в сусідній CCX проходить з майже на порядок меншою швидкістю. Також на кристалі розташовуються два канали DDR4, USB, канали вводу/виводу малої потужності, 4 сервіси IFOP та 2 IFIS, які використовуються для з'єднання декількох кристалів та сокетів разом. [19]

Шина Infinity Fabric складається з двох середовищ зв’язку Infinity Scalable Data Fabric (SDF) та Infinity Scalable Control Fabric (SCF). SDF – це основне середовище для передачі даних від точки до точки. SDF з’єднує між собою PCIe PHY, USB, контролери пам’яті, різні обчислювальні та виконавчі блоки. SCF – це додаткове середовище, що управляє передачею різних системних сигналів управління.

IFOP – 32-бітна шина, що відповідає за комунікацію між кристалами.

IFIS – 16-бітна шина, що використовується для підключення до сокету. IFIS була розроблена таким чином, щоб вона могла мультиплексуватися з іншими протоколами, такими як PCIe та SATA. [20]

В обраному процесорі дані можуть пересилатись напряму від кожного ядра до кожного. Обмеженням є пропускна здатність шини Infinity Fabric і довжина шляху

передачі. Відомо, що передача даних в межах CCX буде швидшою, ніж між сусідніми CCX на одному кристалі. А швидкість передачі даних між кристалами буде повільнішою, за передачу між сусідніми CCX на одному кристалі.

3.2 Вибір мови програмування та програмного забезпечення

Основними критеріями для вибору мови програмування стала швидкість виконання коду, обчислювальна продуктивність та швидкість розробки програми. Саме тому для розробки програмного забезпечення для ПАК було обрано мову C++. В C++ швидке (іноді дуже швидке) виконання коду в порівнянні з більш високорівневими мовами (Python, C#, Java та іншими). Ця мова спроектована так, щоб дати програмісту максимальний контроль над усіма аспектами структури і порядком виконання програми. Один з базових принципів C++ «не плати за те, що не використовуєш» (з філософії C++) – тобто жодна з можливостей, що призводить до додаткових накладних витрат, не є обов'язковою для використання. Є можливість роботи з пам'яттю на низькому рівні. Немає «віртуальних машин» чи фреймворків, які займаються, наприклад, збиранням сміття або виділенням пам'яті. Є повний доступ до API операційної системи без обгорток (в яких може бути реалізовано не все).

Основою кластера є комунікаційне середовище (PVM, MPI), що забезпечує можливість частинам паралельної програми, що виконуються на різних комп'ютерах, ефективно взаємодіяти між собою. Обидва середовища використовують метод передачі повідомлень для комунікації між процесорами.

PVM - це безкоштовне програмне забезпечення (випущене як за ліцензією BSD, так і за загальною публічною ліцензією GNU), що дозволяє обмінюватися даними між процесорами. PVM розробляється і підтримується в рамках дослідницького проекту з 1989 р. В даний час в розвитку проекту беруть участь багато фахівців зі всьому світі. Основна мета - забезпечення роботи паралельних програм в гетерогенних обчислювальних середовищах. В основу PVM покладена концепція віртуальної паралельної обчислювальної машини, яка об'єднує в єдиний ресурс безліч (до тисяч) різнорідних вузлів. Віртуальна машина допускає динамічне конфігурування (причому користувачем, а не адміністратором). Це є однією з причин наявності

механізму асинхронних повідомлень про значимі події в віртуальній машині (додавання/видалення вузла, вихід вузла з ладу).

MPI - це широковідомий стандарт передачі повідомлень, який визначає синтаксис та семантику ядра підпрограм бібліотеки для обміну даними між процесорами. Тому MPI - це не сама технологія, а лише визначення способів обміну даними. Однак реалізація MPI існує в декількох мовах програмування (наприклад, C, C++, FORTRAN, Python і R), і такі реалізації зазвичай називаються теж MPI. Цей факт може викликати плутанину, оскільки і технологія, і визначення стандартів мають одне ім'я. Основна мета стандарту - досягнення максимальної продуктивності (навіть на шкоду гнучкості) виконання паралельних програм на супер-ЕОМ і максимальному ступені переносимості таких програм на рівні вихідних текстів. MPI характеризується статичної структурою обчислювальної середовища, що задається до початку виконання паралельної програми, однак, має засоби динамічного побудови на цій структурі віртуальних топологій обміну і створення груп процесів. Є безліч комерційних реалізацій стандарту MPI для конкретних супер-ЕОМ, існують також вільно поширювані версії, найбільш відома з яких - MPICH (MPI Chameleon), орієнтована, в першу чергу, на використання в кластерах робочих станцій

Порівняння MPI і PVM дозволяє зробити висновок про перевагу використання MPI в однорідних обчислювальних середовищах (машини з масовим паралелізмом і невеликі кластери з однотипних машин).

3.3 Розробка тестових програм

Припустимо, що вхідні дані будуть знаходитись на одному ПК в системі. Тоді розсилка даних буде складатись з таких етапів:

1) Передача на інші ПК в системі.
2) Передача на кожен кристал в межах одного процесора.
3) Передача в межах кожного кристала на CCX.
4) Розсилка в межах CCX.

Кожен CCX складається з 4 процесорів, на яких може виконуватись 8 потоків одночасно. Всього в системі може виконуватись 128 потоків одночасно. На ПК1 це

будуть потоки T0-T31, на ПК2 – T32-T63, на ПК3 – T64-T95, на ПК4 – T96-T127. Відповідно по 16 потоків на CCD, по 8 на CCX. Розглянемо три приклади.

3.3.1 Задача 1

Задача: MA = MB*MC, де MB і MC – матриці розмірності n x n.

Паралельний алгоритм задачі:

1) $MA_H = MB*MC_H$

Спільний ресурс: MB.

Розробка алгоритмів потоків

T0

1	Введення MB, MC
2	Передача MB, MC_{32H} в T32
3	Передача MB, MC_{32H} в T64
4	Передача MB, MC_{32H} в T96
5	Передача MB, MC_{16H} в T16
6	Передача MB, MC_{8H} в T8
7	Передача MB, MC_{4H} в T(id+4)
8	Передача MB, MC_H в T(id+1)
9	Передача MB, MC_H в T(id+2)
10	Передача MB, MC_H в T(id+3)
11	Обчислення $MA_H = MB*MC_H$
12	Отримання MA_H від T(id+1)
13	Отримання MA_H від T(id+2)
14	Отримання MA_H від T(id+3)
15	Отримання MA_{4H} від T(id+4)
16	Отримання MA_{8H} від T(id+8)
17	Отримання MA_{16H} від T(id+16)
18	Отримання MA_{32H} від T32
19	Отримання MA_{32H} від T64
20	Отримання MA_{32H} від T96
21	Вивід MA

T32, T64, T96

1 Отримання MB, MC_{32H} від T0

2 Передача MB, MC_{16H} в T(id+16)

3 Передача MB, MC_{8H} в T(id+8)

4 Передача MB, MC_{4H} в T(id+4)

5 Передача MB, MC_{H} в T(id+1)

6 Передача MB, MC_{H} в T(id+2)

7 Передача MB, MC_{H} в T(id+3)

8 Обчислення MA_{H} = MB*MC_{H}

9 Отримання MA_{H} від T(id+1)

10 Отримання MA_{H} від T(id+2)

11 Отримання MA_{H} від T(id+3)

12 Отримання MA_{4H} від T(id+4)

13 Отримання MA_{8H} від T(id+8)

14 Отримання MA_{16H} від T(id+16)

15 Передача MA_{32H} в T0

T16, T48, T80, T112

1 Отримання MB, MC_{16H} від T(id – 16)

2 Передача MB, MC_{8H} в T(id+8)

3 Передача MB, MC_{4H} в T(id+4)

4 Передача MB, MC_{H} в T(id+1)

5 Передача MB, MC_{H} в T(id+2)

6 Передача MB, MC_{H} в T(id+3)

7 Обчислення MA_{H} = MB*MC_{H}

8 Отримання MA_{H} від T(id+1)

9 Отримання MA_{H} від T(id+2)

10 Отримання MA_{H} від T(id+3)

11 Отримання MA_{4H} від T(id+4)

12 Отримання MA_{8H} від T(id+8)

13 Передача MA_{16H} в T(id – 16)

T8, T24, T40, T56, T72, T88, T104, T120

1 Отримання MB, MC_{8H} від T(id – 8)

2 Передача MB, MC_{4H} в T(id+4)

3 Передача MB, MC_H в T(id+1)

4 Передача MB, MC_H в T(id+2)

5 Передача MB, MC_H в T(id+3)

6 Обчислення MA_H = MB*MC_H

7 Отримання MA_H від T(id+1)

8 Отримання MA_H від T(id+2)

9 Отримання MA_H від T(id+3)

10 Отримання MA_{4H} від T(id+4)

11 Передача MA_{8H} в T(id – 8)

T(4+i*8), i=($\overline{0, 15}$)

1 Отримання MB, MC_{4H} від T(id – 4)

2 Передача MB, MC_H в T(id+1)

3 Передача MB, MC_H в T(id+2)

4 Передача MB, MC_H в T(id+3)

5 Обчислення MA_H = MB*MC_H

6 Отримання MA_H від T(id+1)

7 Отримання MA_H від T(id+2)

8 Отримання MA_H від T(id+3)

9 Передача MA_{4H} в T(id – 4)

Tid % 4 = 1

1 Отримання MB, MC_H від T(id – 1)

2 Обчислення MA_H = MB*MC_H

3 Передача MA_H в T(id – 1)

Tid % 4 = 2

1 Отримання MB, MC_H від T(id – 2)

2 Обчислення MA_H = MB*MC_H

3 Передача MA_H в T(id – 2)

Tid % 4 = 3

1 Отримання MB, MC_H від T(id – 3)

2 Обчислення MA_H = MB*MC_H

3 Передача MA_H в T(id – 3)

Схему взаємодії потоків наведено у додатку В.

Лістинг програми наведено у додатку Е.

3.3.2 Задача 2

Задача: MA = MB*(MC*MD), де MB, MC і MD – матриці розмірності n x n.

Паралельний алгоритм задачі:

1) MT_H = MC*MD_H

2) MA_H = MB*MT_H

Спільні ресурси: MB, MC

Розробка алгоритмів потоків.

T0

1 Введення MB, MC, MD

2 Передача MB, MD, MC_{32H} в T32

3 Передача MB, MD, MC_{32H} в T64

4 Передача MB, MD, MC_{32H} в T96

5 Передача MB, MD, MC_{16H} в T16

6 Передача MB, MD, MC_{8H} в T8

7 Передача MB, MD, MC_{4H} в T(id+4)

8 Передача MB, MD, MC_H в T(id+1)

9 Передача MB, MD, MC_H в T(id+2)

10 Передача MB, MD, MC_H в T(id+3)

11 Обчислення MT_H = MC*MD_H

12 Обчислення $MA_H = MB*MT_H$

13 Отримання MA_H від T(id+1)

14 Отримання MA_H від T(id+2)

15 Отримання MA_H від T(id+3)

16 Отримання MA_{4H} від T(id+4)

17 Отримання MA_{8H} від T(id+8)

18 Отримання MA_{16H} від T(id+16)

19 Отримання MA_{32H} від T32

20 Отримання MA_{32H} від T64

21 Отримання MA_{32H} від T96

22 Вивід MA

T32, T64, T96

1 Отримання MB, MD, MC_{32H} від T1

2 Передача MB, MD, MC_{16H} в T(id+16)

3 Передача MB, MD, MC_{8H} в T(id+8)

4 Передача MB, MD, MC_{4H} в T(id+4)

5 Передача MB, MD, MC_H в T(id+1)

6 Передача MB, MD, MC_H в T(id+2)

7 Передача MB, MD, MC_H в T(id+3)

8 Обчислення $MT_H = MC*MD_H$

9 Обчислення $MA_H = MB*MT_H$

10 Отримання MA_H від T(id+1)

11 Отримання MA_H від T(id+2)

12 Отримання MA_H від T(id+3)

13 Отримання MA_{4H} від T(id+4)

14 Отримання MA_{8H} від T(id+8)

15 Отримання MA_{16H} від T(id+16)

16 Передача MA_{32H} в T0

T16, T48, T80, T112

1 Отримання MB, MD, MC_{16H} від T(id – 16)

2 Передача MB, MD, MC_{8H} в T(id+8)

3 Передача MB, MD, MC_{4H} в T(id+4)

4 Передача MB, MD, MC_{H} в T(id+1)

5 Передача MB, MD, MC_{H} в T(id+2)

6 Передача MB, MD, MC_{H} в T(id+3)

7 Обчислення $MT_{H} = MC*MD_{H}$

8 Обчислення $MA_{H} = MB*MT_{H}$

9 Отримання MA_{H} від T(id+1)

10 Отримання MA_{H} від T(id+2)

11 Отримання MA_{H} від T(id+3)

12 Отримання MA_{4H} від T(id+4)

13 Отримання MA_{8H} від T(id+8)

14 Передача MA_{16H} в T(id – 16)

T8, T24, T40, T56, T72, T88, T104, T120

1 Отримання MB, MD, MC_{8H} від T(id – 8)

2 Передача MB, MD, MC_{4H} в T(id+4)

3 Передача MB, MD, MC_{H} в T(id+1)

4 Передача MB, MD, MC_{H} в T(id+2)

5 Передача MB, MD, MC_{H} в T(id+3)

6 Обчислення $MT_{H} = MC*MD_{H}$

7 Обчислення $MA_{H} = MB*MT_{H}$

8 Отримання MA_{H} від T(id+1)

9 Отримання MA_{H} від T(id+2)

10 Отримання MA_{H} від T(id+3)

11 Отримання MA_{4H} від T(id+4)

12 Передача MA_{8H} в T(id – 8)

T(4+i*8), i=($\overline{0, 15}$)

1 Отримання MB, MD, MC_{4H} від T(id – 4)

2 Передача MB, MD, MC_H в T(id+1)

3 Передача MB, MD, MC_H в T(id+2)

4 Передача MB, MD, MC_H в T(id+3)

5 Обчислення $MT_H = MC*MD_H$

6 Обчислення $MA_H = MB*MT_H$

7 Отримання MA_H від T(id+1)

8 Отримання MA_H від T(id+2)

9 Отримання MA_H від T(id+3)

10 Передача MA_{4H} в T(id – 4)

Tid % 4 = 1

1 Отримання MB, MD, MC_H від T(id – 1)

2 Обчислення $MT_H = MC*MD_H$

3 Обчислення $MA_H = MB*MT_H$

4 Передача MA_H в T(id – 1)

Tid % 4 = 2

1 Отримання MB, MD, MC_H від T(id – 2)

2 Обчислення $MT_H = MC*MD_H$

3 Обчислення $MA_H = MB*MT_H$

4 Передача MA_H в T(id – 2)

Tid % 4 = 3

1 Отримання MB, MD, MC_H від T(id – 3)

2 Обчислення $MT_H = MC*MD_H$

3 Обчислення $MA_H = MB*MT_H$

4 Передача MA_H в T(id – 3)

Схему взаємодії потоків наведено у додатку Г.

Лістинг програми наведено у додатку Є.

3.3.3 Задача 3

Задача: A = B*MC+D*ME, де MC і ME – матриці розмірності n x n, а B і D вектори з n елементів

Паралельний алгоритм задачі:

1) $A1_H = B*MC_H$
2) $A2_H = D*ME_H$
3) $A_H = A1_H + A2_H$

Спільні ресурси: B, D

Розробка алгоритмів потоків

T0

1 Введення MC, ME, B, D
2 Передача B, D, MC_{32H}, ME_{32H} в T32
3 Передача B, D, MC_{32H}, ME_{32H} в T64
4 Передача B, D, MC_{32H}, ME_{32H} в T96
5 Передача B, D, MC_{16H}, ME_{16H} в T16
6 Передача B, D, MC_{8H}, ME_{8H} в T8
7 Передача B, D, MC_{4H}, ME_{4H} в T4
8 Передача B, D, MC_H, ME_H в T1
9 Передача B, D, MC_H, ME_H в T2
10 Передача B, D, MC_H, ME_H в T3
11 Обчислення $A1_H = B*MC_H$
12 Обчислення $A2_H = D*ME_H$
13 Обчислення $A_H = A1_H + A2_H$
14 Отримання A_H від T(id+1)
15 Отримання A_H від T(id+2)
16 Отримання A_H від T(id+3)
17 Отримання A_{4H} від T(id+4)
18 Отримання A_{8H} від T(id+8)
19 Отримання A_{16H} від T(id+16)
20 Отримання A_{32H} від T32

21 Отримання A_{32H} від T64

22 Отримання A_{32H} від T96

23 Вивід A

T32, T64, T96

1 Отримання B, D, MC_{32H}, ME_{32H} від T0

2 Передача B, D, MC_{16H}, ME_{16H} в T(id+16)

3 Передача B, D, MC_{8H}, ME_{8H} в T(id+8)

4 Передача B, D, MC_{4H}, ME_{4H} в T(id+4)

5 Передача B, D, MC_H, ME_H в T(id+1)

6 Передача B, D, MC_H, ME_H в T(id+2)

7 Передача B, D, MC_H, ME_H в T(id+3)

8 Обчислення $A1_H = B*MC_H$

9 Обчислення $A2_H = D*ME_H$

10 Обчислення $A_H = A1_H + A2_H$

11 Отримання A_H від T(id+1)

12 Отримання A_H від T(id+2)

13 Отримання A_H від T(id+3)

14 Отримання A_{4H} від T(id+4)

15 Отримання A_{8H} від T(id+8)

16 Отримання A_{16H} від T(id+16)

17 Передача A_{32H} в T0

T16, T48, T80, T112

1 Отримання B, D, MC_{16H}, ME_{16H} від T(id – 16)

2 Передача B, D, MC_{8H}, ME_{8H} в T(id+8)

3 Передача B, D, MC_{4H}, ME_{4H} в T(id+4)

4 Передача B, D, MC_H, ME_H в T(id+1)

5 Передача B, D, MC_H, ME_H в T(id+2)

6 Передача B, D, MC_H, ME_H в T(id+3)

7 Обчислення $A1_H = B*MC_H$

8 Обчислення $A2_H = D*ME_H$

9 Обчислення $A_H = A1_H + A2_H$

10 Отримання A_H від T(id+1)

11 Отримання A_H від T(id+2)

12 Отримання A_H від T(id+3)

13 Отримання A_{4H} від T(id+4)

14 Отримання A_{8H} від T(id+8)

15 Передача A_{16H} в T(id – 16)

T8, T24, T40, T56, T72, T88, T104, T120

1 Отримання B, D, MC_{8H}, ME_{8H} від T(id – 8)

2 Передача B, D, MC_{4H}, ME_{4H} в T(id+4)

3 Передача B, D, MC_H, ME_H в T(id+1)

4 Передача B, D, MC_H, ME_H в T(id+2)

5 Передача B, D, MC_H, ME_H в T(id+3)

6 Обчислення $A1_H = B*MC_H$

7 Обчислення $A2_H = D*ME_H$

8 Обчислення $A_H = A1_H + A2_H$

9 Отримання A_H від T(id+1)

10 Отримання A_H від T(id+2)

11 Отримання A_H від T(id+3)

12 Отримання A_{4H} від T(id+4)

13 Передача A_{8H} в T(id – 8)

T(4+i*8), $i=\overline{(0, 15)}$

1 Отримання B, D, MC_{4H}, ME_{4H} від T(id – 4)

2 Передача B, D, MC_H, ME_H в T(id+1)

3 Передача B, D, MC_H, ME_H в T(id+2)

4 Передача B, D, MC_H, ME_H в T(id+3)

5 Обчислення $A1_H = B*MC_H$

6 Обчислення $A2_H = D*ME_H$

7 Обчислення $A_H = A1_H + A2_H$

8 Отримання A_H від T(id+1)

9 Отримання A_H від T(id+2)

10 Отримання A_H від T(id+3)

11 Передача A_{4H} в T(id – 4)

Tid % 4 = 1

1 Отримання B, D, MC_H, ME_H від T(id – 1)

2 Обчислення $A1_H = B*MC_H$

3 Обчислення $A2_H = D *ME_H$

4 Обчислення $A_H = A1_H + A2_H$

5 Передача A_H в T(id – 1)

Tid % 4 = 2

1 Отримання B, D, MC_H, ME_H від T(id – 2)

2 Обчислення $A1_H = B*MC_H$

3 Обчислення $A2_H = D *ME_H$

4 Обчислення $A_H = A1_H + A2_H$

5 Передача A_H в T(id – 2)

Tid % 4 = 3

1 Отримання B, D, MC_H, ME_H від T(id – 3)

2 Обчислення $A1_H = B*MC_H$

3 Обчислення $A2_H = D *ME_H$

4 Обчислення $A_H = A1_H + A2_H$

5 Передача A_H в T(id – 3)

Схему взаємодії потоків наведено у додатку Д.

Лістинг програми наведено у додатку Ж.

ВИСНОВКИ ДО РОЗДІЛУ 3

1. В даному розділі було розроблено ПАК для матричних операцій.
2. Було обрано три задачі з використанням матричних операцій, що часто використовуються для обчислень.
3. Було розроблено паралельні алгоритми для цих задач, алгоритми виконання потоків.
4. Було розроблено три програми за написаними алгоритмами.

РОЗДІЛ 4. ТЕСТУВАННЯ РОЗРОБЛЕНОГО ПРОГРАМНОГО ЗАБЕПЕЧЕННЯ ДЛЯ ПАК.

Для дослідження необхідно визначити час виконання розроблених програм в реальній КС. Програми тестувались на AMD FX-6300 CPU @ 4.10GHz , що має 6 ядер з 16Гб ОП. При тестуванні послідовно використовуються 1, 2, 4, 6 ядер процесора, для яких визначається час виконання задач. При цьому встановлюється декілька значень розмірності матриць (векторів). Підрахували час виконання для всіх задач при різних параметрах N та P.

Для оцінки розроблених програм обрахували значення коефіцієнту прискорення та коефіцієнту ефективності.

Підрахунок коефіцієнту прискорення (КП) та коефіцієнту ефективності (КЕ) відбувається за формулами:

$$КП = T1 / Tp; \qquad КЕ = КП / P * 100\%.$$

4.1 Тест задачі 1

Із збільшенням кількості обчислювальних вузлів, час роботи програми зменшується, як показано у таблиці 4.1.

Таблиця 4.1 Час виконання задачі 1 (значення в секундах)

N	Кількість задіяних ядер (P)			
	1	2	4	6
1280	354.02	203.21	103.39	68.91
1920	564.86	316.70	158.49	104.80
2560	808.50	447.92	220.86	144.49
3200	1081.28	585.89	286.14	187.23

Значення КП лежать в межах від 1.74 до 5.78, як показано в таблиці 4.2. Максимальне значення КП = 5.78 отримано при P = 6, N = 3200. Мінімальне значення КП = 1.74 отримано при P = 2, N = 1280. Динаміка зміни КП показана на рис. 4.1.

Значення КЕ змінюються від 86% до 96%, як показано в таблиці 4.3. Максимальне значення КЕ = 96% отримано при P = 6, N = 3200. Мінімальне значення КЕ = 86% отримано при P = 4, N = 1280. Динаміка зміни КП показана на рис. 4.2.

Таблиця 4.2 Значення КП для задачі 1

N	Кількість задіяних ядер (P)			
	1	2	4	6
1280	1.00	1.74	3.42	5.14
1920	1.00	1.78	3.56	5.39
2560	1.00	1.81	3.66	5.60
3200	1.00	1.85	3.78	5.78

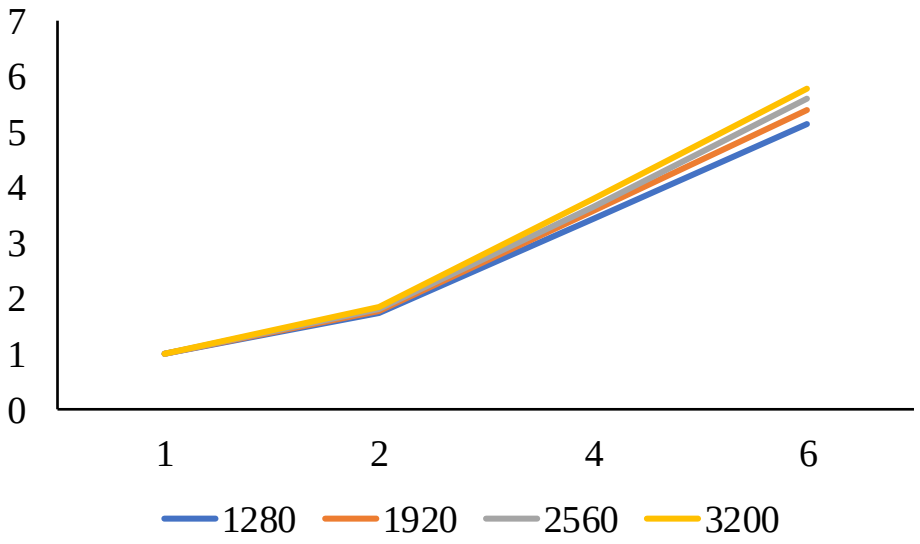


Рис 4.1. Графік залежності коефіцієнту прискорення від кількості процесорів для задачі 1

Таблиця 4.3 Значення КЕ для задачі 1

N	Кількість задіяних ядер (P)			
	1	2	4	6
1280	100%	87%	86%	86%
1920	100%	89%	89%	90%
2560	100%	90%	92%	93%
3200	100%	92%	94%	96%

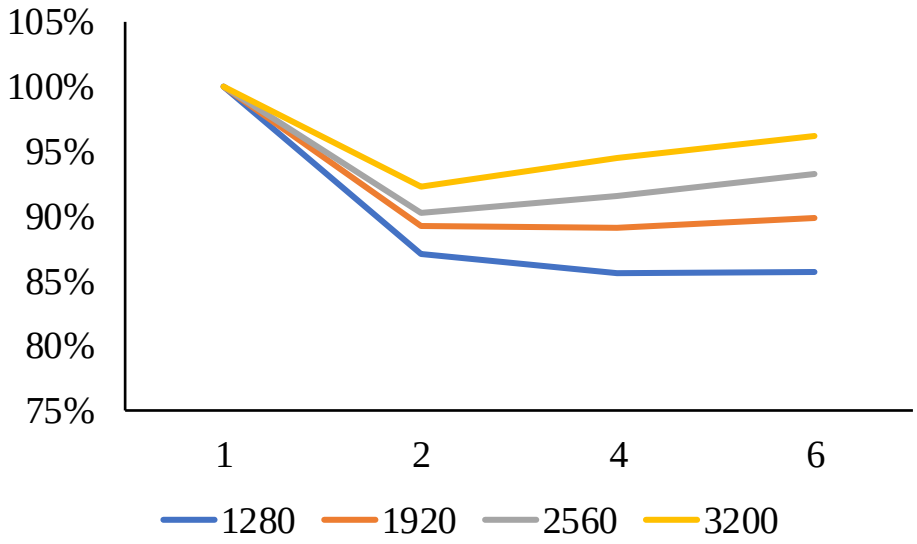


Рис 4.2. Графік залежності коефіцієнту ефективності від кількості процесорів для задачі 1

4.2 Тест задачі 2

Із збільшенням кількості обчислювальних вузлів, час роботи програми зменшується, як показано у таблиці 4.4.

Таблиця 4.4 Час виконання задачі 2 (значення в секундах)

N	Кількість задіяних ядер (P)			
	1	2	4	6
1280	715.27	393.06	195.38	128.48
1920	1139.63	618.24	308.10	199.47
2560	1598.63	860.95	427.57	276.96
3200	2216.73	1187.11	583.14	379.97

Значення КП лежать в межах від 1.82 до 5.83, як показано в таблиці 4.5. Максимальне значення КП = 5.83 отримано при P = 6, N = 3200. Мінімальне значення КП = 1.82 отримано при P = 2, N = 1280. Динаміка зміни КП показана на рис. 4.3.

Значення КЕ змінюються від 91% до 97%, як показано в таблиці 4.6. Максимальне значення КЕ = 97% отримано при P = 6, N = 3200. Мінімальне значення КЕ = 91% отримано при P = 2, N = 1280. Динаміка зміни КП показана на рис. 4.4.

Таблиця 4.5 Значення КП для задачі 2

N	Кількість задіяних ядер (P)			
	1	2	4	6
1280	1.00	1.82	3.66	5.57
1920	1.00	1.84	3.70	5.71
2560	1.00	1.86	3.74	5.77
3200	1.00	1.87	3.80	5.83

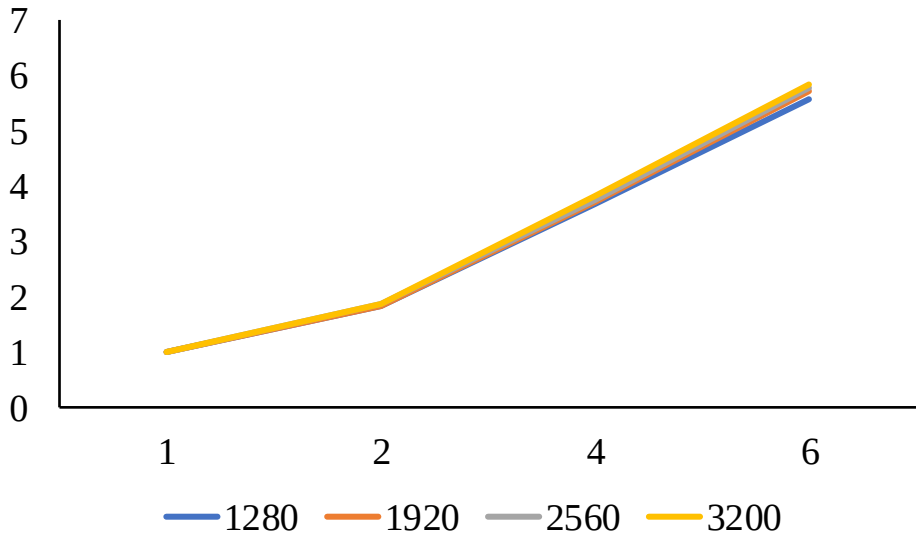


Рис 4.3 Графік залежності коефіцієнту прискорення від кількості процесорів для задачі 2

Таблиця 4.6 Значення КЕ для задачі 2

N	Кількість задіяних ядер (P)			
	1	2	4	6
1280	100%	91%	92%	93%
1920	100%	92%	92%	95%
2560	100%	93%	93%	96%
3200	100%	93%	95%	97%

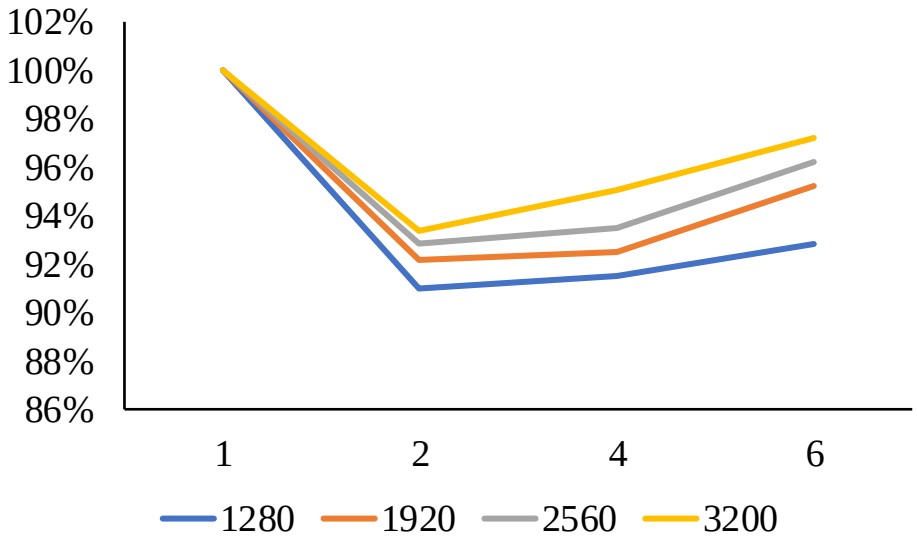


Рис 4.4 Графік залежності коефіцієнту ефективності від кількості процесорів для задачі 2

4.3 Тест задачі 3

Із збільшенням кількості обчислювальних вузлів час роботи програми зменшується, як показано у таблиці 4.7.

Таблиця 4.7 Час виконання задачі 3 (значення в секундах)

N	Кількість задіяних ядер (P)			
	1	2	4	6
1280	1.33	0.78	0.38	0.24
2560	7.68	4.41	2.16	1.39
5120	107.22	59.81	29.36	18.71
10240	441.26	243.95	119.39	75.05

Значення КП лежать в межах від 1.72 до 5.88, як показано в таблиці 4.8. Максимальне значення КП = 5.88 отримано при P = 6, N = 10240. Мінімальне значення КП = 1.72 отримано при P = 2, N = 1280. Динаміка зміни КП показана на рис. 4.5.

Значення КЕ змінюються від 86% до 98%, як показано в таблиці 4.9. Максимальне значення КЕ = 98% отримано при P = 6, N = 10240. Мінімальне значення КЕ = 86% отримано при P = 2, N = 1280. Динаміка зміни КП показана на рис. 4.6.

Таблиця 4.8 Значення КП для задачі 3

N	Кількість задіяних ядер (P)			
	1	2	4	6
1280	1.00	1.72	3.54	5.51
2560	1.00	1.74	3.55	5.54
5120	1.00	1.79	3.65	5.73
10240	1.00	1.81	3.70	5.88

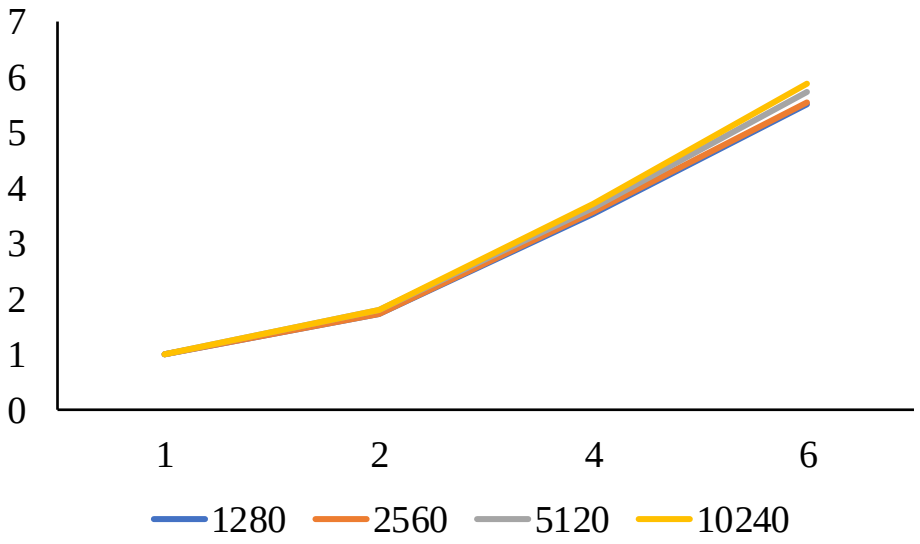


Рис 4.5 Графік залежності коефіцієнту прискорення від кількості процесорів для задачі 3

Таблиця 4.9 Значення КЕ для задачі 3

N	Кількість задіяних ядер (P)			
	1	2	4	6
1280	100%	86%	88%	92%
2560	100%	87%	89%	92%
5120	100%	90%	91%	96%
10240	100%	90%	92%	98%

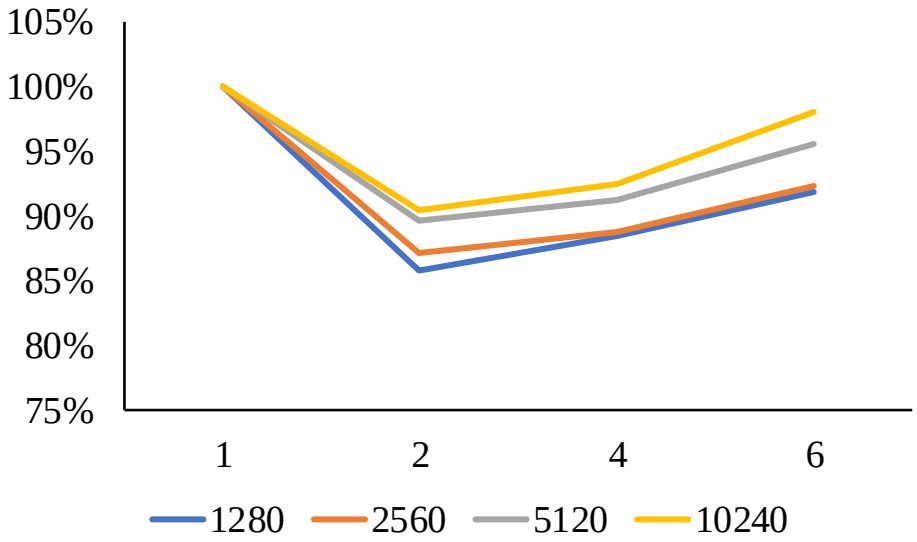


Рис 4.6 Графік залежності коефіцієнту ефективності від кількості процесорів для задачі 3

Загрузка ядер процесора при тестуванні показано на рис. 4.7-4.10.

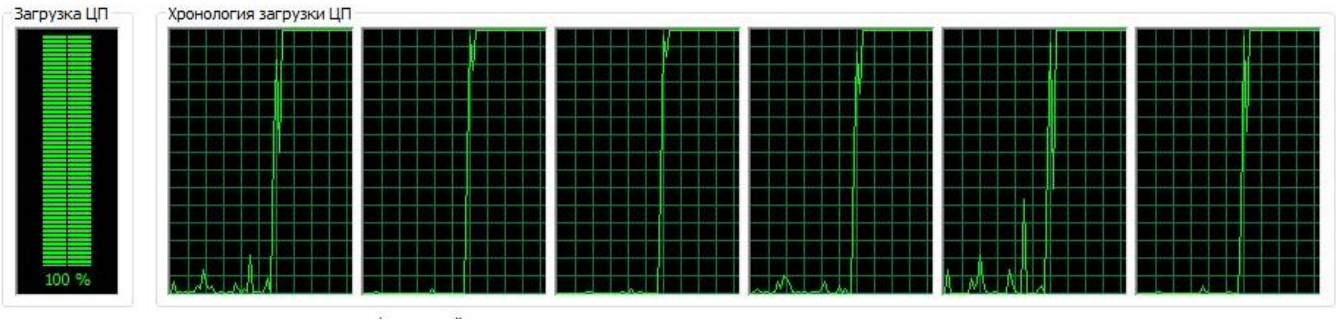


Рис. 4.7 Загрузка процесора при тестуванні на 6 ядрах.

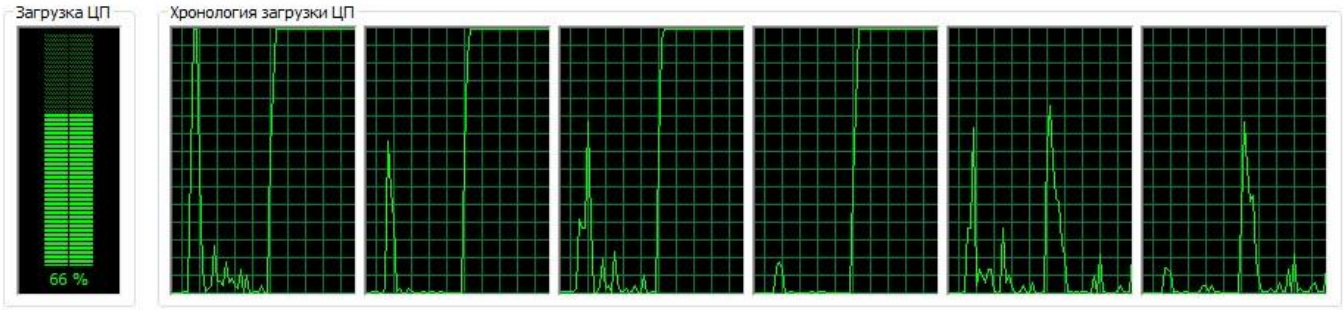


Рис. 4.8 Загрузка процесора при тестуванні на 4 ядрах.

Рис. 4.9 Загрузка процесора при тестуванні на 2 ядрах.

Рис. 4.10 Загрузка процесора при тестуванні на 1 ядрі.

ВИСНОВКИ ДО РОЗДІЛУ 4

1. В даному розділі було протестовано програми для ПАК на процесорі AMD FX-6300 CPU @ 4.10GHz , що має 6 ядер та з 16Гб ОП.
2. Тестування показало, що розроблені програми здатні використати обчислювальні потужності системи на 100%.
3. Всі програми дають прискорення при виконанні матричних операцій.
4. З ростом розмірності матриць (векторів) КП зростає для всіх значень.

ЛІТЕРАТУРА

1. Intel® Xeon® Scalable processors [Електронний ресурс] – режим доступу: https://www.intel.com/content/www/us/en/products/docs/processors/xeon/ultimate-workstation-performance.html
2. Zen 2 - Microarchitecture - AMD [Електронний ресурс] – режим доступу: https://www.amd.com/ru/technologies/zen-core
3. Порівняння продукції Intel [Електронний ресурс] – режим доступу: https://ark.intel.com/content/www/ru/ru/ark/compare.html?productIds=126699,126697,189126,189123,198017
4. Порівняння продукції Intel [Електронний ресурс] – режим доступу: https://www.intel.ru/content/www/ru/ru/products/compare-products.html/processors?productIds=192943,191044,191789,186605
5. AMD Ryzen Threadripper 3d Gen [Електронний ресурс] – режим доступу: https://www.amd.com/ru/press-releases/2019-11-07-amd-introduces-world-s-fastest-high-end-desktop-processors-3rd-gen-ryzen
6. AMD Ryzen Threadripper 3990X [Електронний ресурс] – режим доступу: https://www.amd.com/en/products/cpu/amd-ryzen-threadripper-3990x
7. AMD Ryzen Threadripper 2990WX [Електронний ресурс] – режим доступу: https://www.amd.com/ru/products/cpu/amd-ryzen-threadripper-2990wx
8. AMD Ryzen Threadripper 2970WX [Електронний ресурс] – режим доступу: https://www.amd.com/ru/products/cpu/amd-ryzen-threadripper-2970wx
9. AMD Ryzen Threadripper 2950X [Електронний ресурс] – режим доступу: https://www.amd.com/ru/products/cpu/amd-ryzen-threadripper-2950x
10. Таненбаум Э. Распределенные системы. Принципы и парадигмы/Э. Таненбаум, М. Ван Стеен. – изд. «Питер», 2003. - 877 с.
11. Антонюк В. А. Спецкурс «Программирование на видеокартах (GPGPU)» [Електронний ресурс] / В. А. Антонюк [Електронний ресурс] – режим доступу: http://cmp.phys.msu.su/sites/default/files/GPGPU.pdf

12. Порівняння характеристик відеокарт [Електронний ресурс] – https://www.nix.ru/price/compare_goods.html?goods_for_compare=428058,418125,394321,435982,436892,435987,447595#tab=compare-properties
13. Жуков І.А., Корочкін О.В. Паралельні та розподілені обчислення: Навч. Посібник / І. А. Жуков, О. В. Корочкін, 2005. – 226 с.
14. Документація Microsoft: ManualResetEvent Class [Електронний ресурс] – режим доступу: https://docs.microsoft.com/en-us/dotnet/api/system.threading.manualresetevent?view=netframework-4.8
15. Перші кроки WinAPI [Електронний ресурс] – режим доступу: http://www.firststeps.ru/mfc/winapi/
16. Курзэнков О. Багатопоточність та синхронізація. Частина 3. Об'єкти синхронізації [Електронний ресурс] / О. Курзэнков [Електронний ресурс] – режим доступу: http://www.kurzenkov.com/Articles/multithreading3.html
17. Руководство по программированию высокопроизводительных вычислительных систем / В. Н.Дацюк, О. В. Дацюк, А. А. Букатов, С. А. Виноградова. – Ростов-на-Дону.
18. Гавва О. О. Адское программирование / О. О. Гавва.
19. Zeppelin – AMD [Електронний ресурс] – режим доступу: https://en.wikichip.org/wiki/amd/zeppelin
20. Infinity Fabric – AMD [Електронний ресурс] – режим доступу: https://en.wikichip.org/wiki/amd/infinity_fabric#Overview

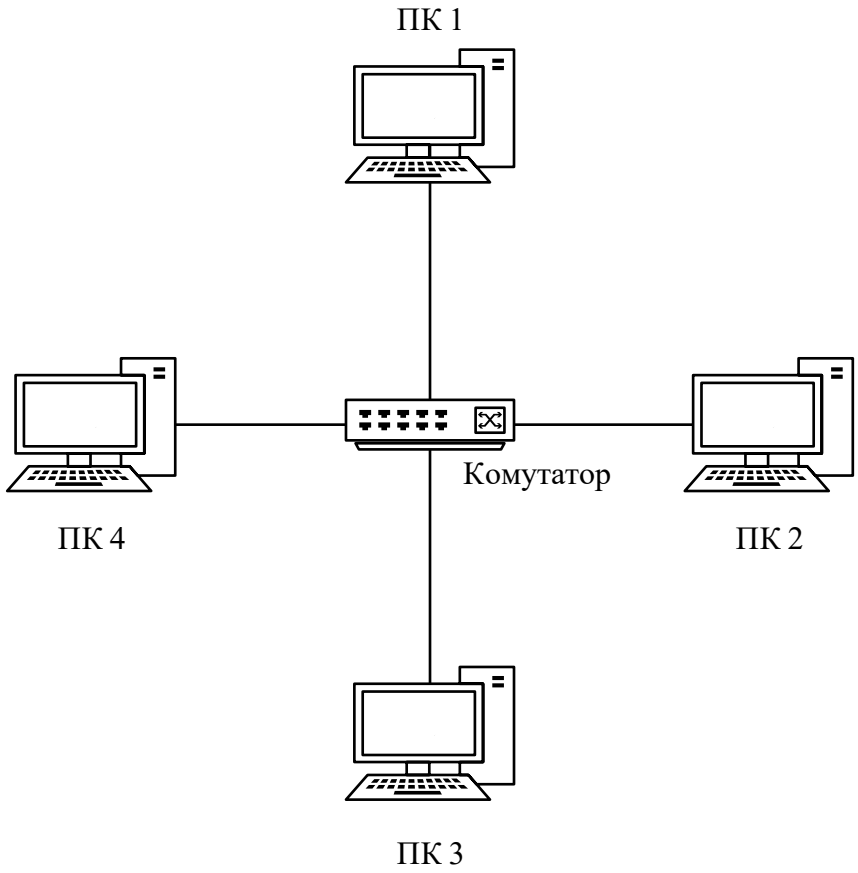
ДОДАТОК А

Програмно-апаратний комплекс для матричних операцій

Схема топології програмно-апаратного комплексу

Аркушів 1

Київ – 2020 року



					ДП 4665.04.000 Д1	
Змн	Арк	№ докум	Підпис	Дата		
Розроб		Бровченко А.В.			*Програмно-апаратний комплекс для матричних операцій* *Схема топології ПАК.*	Літ / Аркуш 1 / Аркушів 1
Перевірив		Корочкін О.В.				
Н. Контр		Симоненко В.П.				НТУУ “КПІ”, ФІОТ, каф. ОТ, гр. ІО-64
Затверд.		Корочкін О.В.				

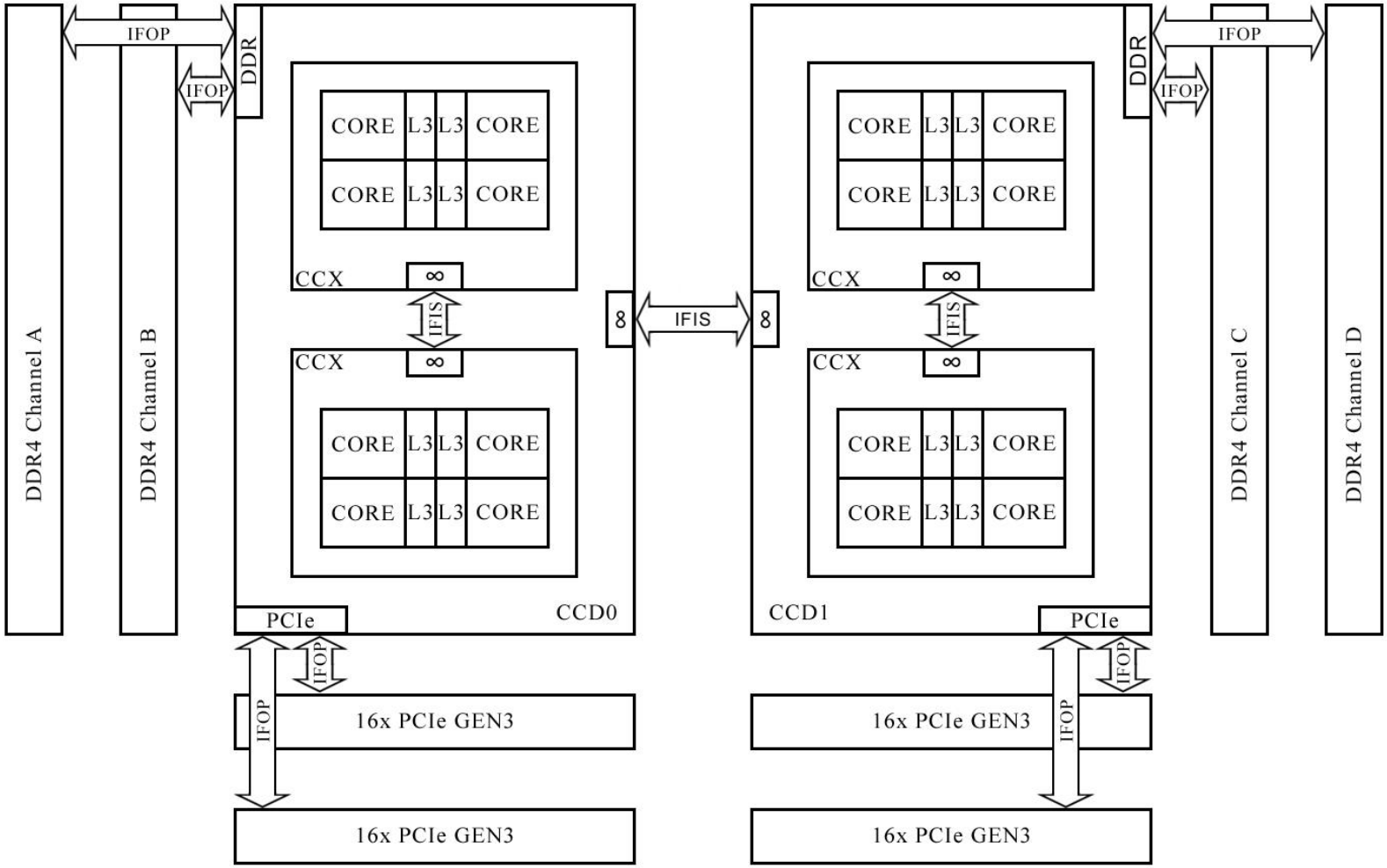
ДОДАТОК Б

Програмно-апаратний комплекс для матричних операцій

Схема структурна процесора AMD 2950X

Аркушів 1

Київ – 2020 року



					ДП 4665.05.000 Д2
Змн	Арк	№ докум	Підпис	Дата	
Розроб		Бровченко А.В.			*Програмно-апаратний комплекс для матричних операцій* *Схема структурна процесора AMD 2950X.*
Перевірив		Корочкін О.В.			
Н. Контр		Симоненко В.П.			
Затверд.		Корочкін О.В.			

Літ	Аркуш	Аркушів
	1	1

НТУУ “КПІ”, ФІОТ, каф. ОТ, гр. ІО-64

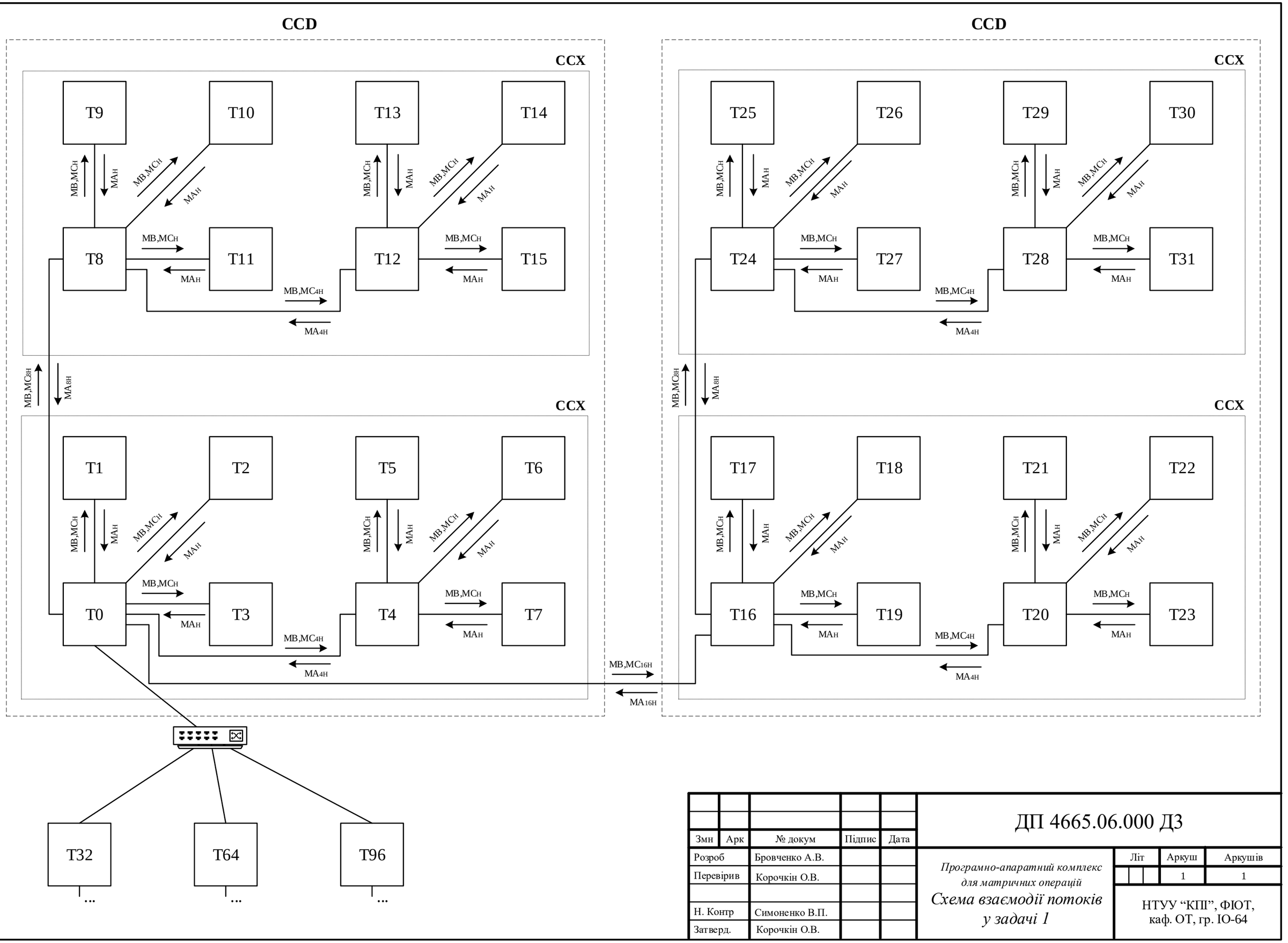
ДОДАТОК В

Програмно-апаратний комплекс для матричних операцій

Схема взаємодії потоків у задачі 1

Аркушів 1

Київ – 2020 року



					ДП 4665.06.000 Д3
Змн	Арк	№ докум	Підпис	Дата	
Розроб		Бровченко А.В.			*Програмно-апаратний комплекс для матричних операцій* *Схема взаємодії потоків у задачі 1*
Перевірив		Корочкін О.В.			
Н. Контр		Симоненко В.П.			
Затверд.		Корочкін О.В.			

Літ	Аркуш	Аркушів
	1	1

НТУУ “КПІ”, ФІОТ, каф. ОТ, гр. ІО-64

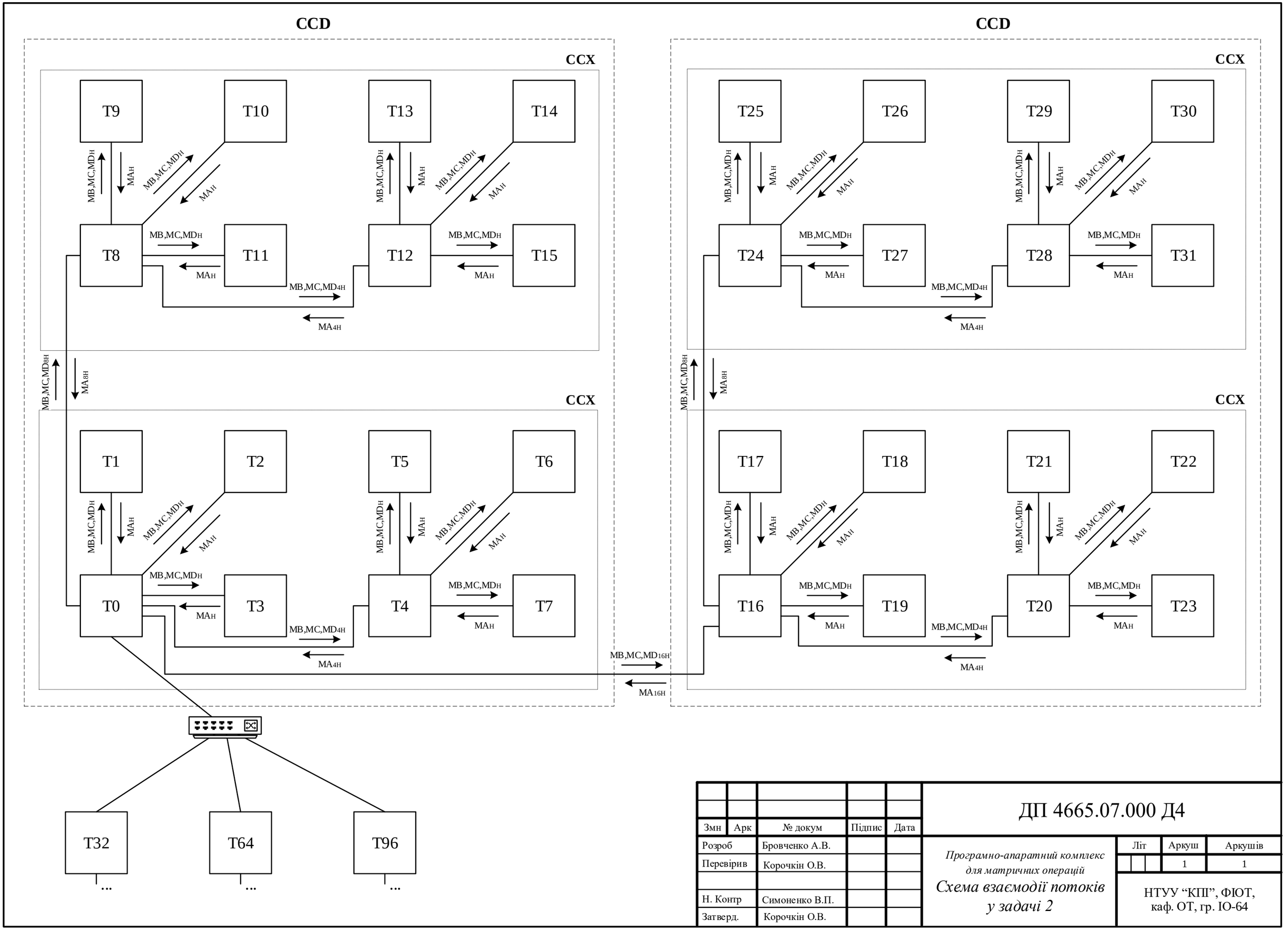
ДОДАТОК Г

Програмно-апаратний комплекс для матричних операцій

Схема взаємодії потоків у задачі 2

Аркушів 1

Київ – 2020 року



					ДП 4665.07.000 Д4
Змн	Арк	№ докум	Підпис	Дата	
Розроб		Бровченко А.В.			*Програмно-апаратний комплекс для матричних операцій* *Схема взаємодії потоків у задачі 2*
Перевірив		Корочкін О.В.			
Н. Контр		Симоненко В.П.			
Затверд.		Корочкін О.В.			

Літ	Аркуш	Аркушів
	1	1

НТУУ “КПІ”, ФІОТ, каф. ОТ, гр. ІО-64

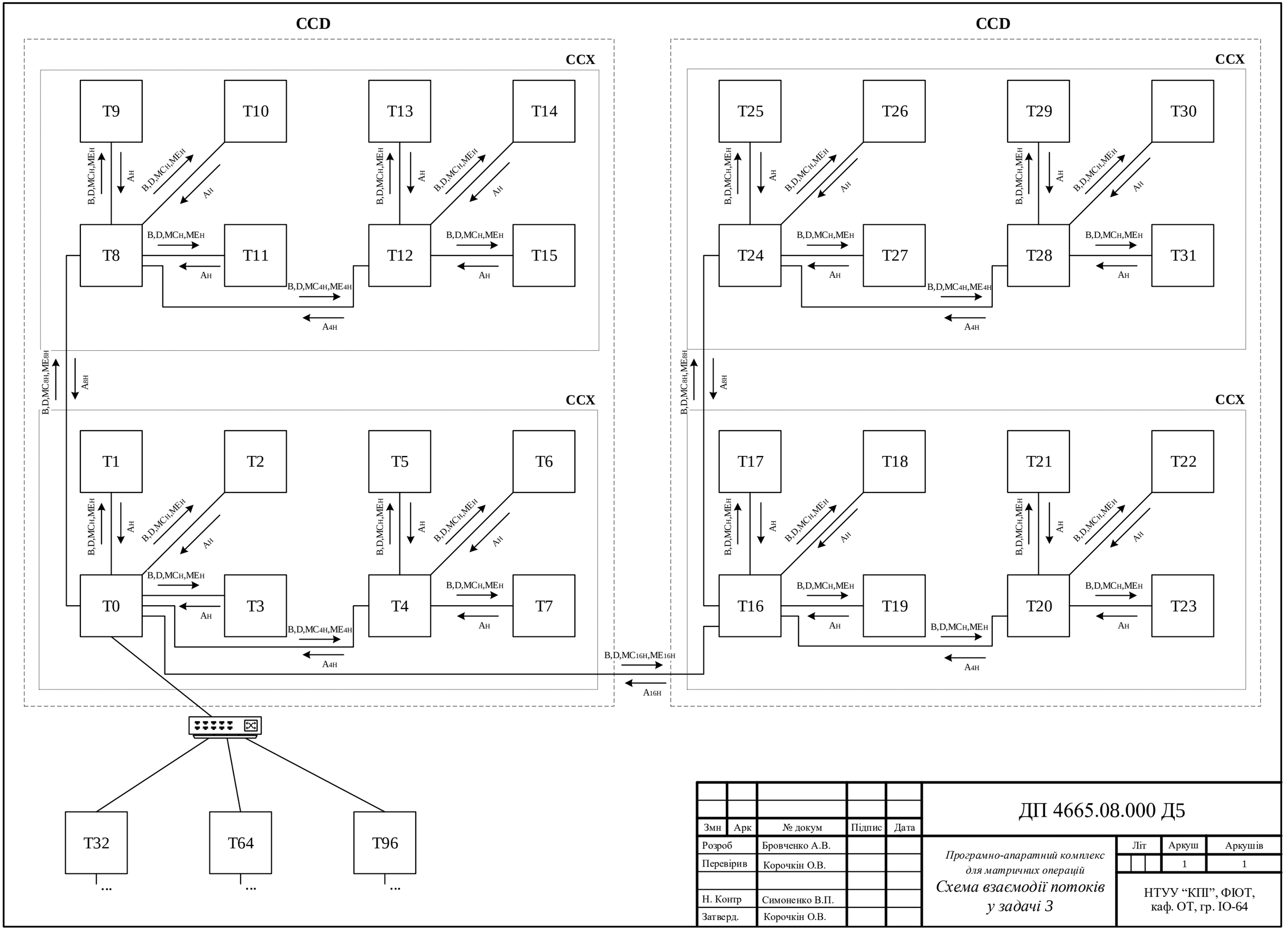
ДОДАТОК Д

Програмно-апаратний комплекс для матричних операцій

Схема взаємодії потоків у задачі 3

Аркушів 1

Київ – 2020 року



					ДП 4665.08.000 Д5
Змн	Арк	№ докум	Підпис	Дата	
Розроб		Бровченко А.В.			*Програмно-апаратний комплекс для матричних операцій* *Схема взаємодії потоків у задачі 3*
Перевірив		Корочкін О.В.			
Н. Контр		Симоненко В.П.			
Затверд.		Корочкін О.В.			

Літ	Аркуш	Аркушів
	1	1

НТУУ “КПІ”, ФІОТ, каф. ОТ, гр. ІО-64

ДОДАТОК Е

Програмно-апаратний комплекс для матричних операцій

Лістинг задачі 1

Аркушів 9

Київ – 2020 року

```
/*
Brovchenko Anastasiya
IO-64
Diploma work. MPI.
MA = MB*MC
10.05.2020
*/
#include "windows.h"
#include <iostream>
#include <fstream>
#include <string>
#include <mpi.h>
#include <ctime>
using namespace std;

//increase stack capathity
#pragma comment(linker, "/STACK:16000")

const int N = 3200;
const int P = 128;
const int H = N / P;
void matrixFillOnes(int* matrix[N]);

int main(int argc, char** argv)
{
	HANDLE process = GetCurrentProcess();
	DWORD_PTR processAffinityMask = 63;
	SetProcessAffinityMask(process, processAffinityMask);
	int rank, size;
	MPI_Status status;
	MPI_Init(&argc, &argv);
	MPI_Comm_size(MPI_COMM_WORLD, &size);
	MPI_Comm_rank(MPI_COMM_WORLD, &rank);

	unsigned int t_start = clock();
	cout << "Task" << rank << " start " << endl;

	if (rank == 0) {
		int** MA = new int* [N];
		int** MB = new int* [N];
		int** MC = new int* [N];

		matrixFillOnes(MB);
		matrixFillOnes(MC);

		// send to 32, 64, 96
		for (int i = 1; i < 4; i++) {
			for (int j = 0; j < N; j++) {
				MPI_Send(MB[j], N, MPI_INTEGER, 32 * i, 0, MPI_COMM_WORLD);
				MPI_Send(MC[j] + i * 32 * H, 32 * H, MPI_INTEGER, 32 * i, 0, MPI_COMM_WORLD);
			}
		}
```

					ДП 4665.09.000 Д6		
Зм.	Арк.	№ докум.	Підпис	Дата			
Розробила		Бровченко А.В.			*Програмно-апаратний комплекс для матричних операцій* *Додаток 6* *Лістинг задачі 1*	Літ.	Аркуш 1 / Аркушів 3
Перевір.		Корочкін О.В.					
Н. контр.		Симоненко В.П.				НТУУ "КПІ", ФІОТ, каф. ОТ, гр. ІО-64	
Затверд.		Корочкін О.В.					

```
                // send to 16
                for (int j = 0; j < N; j++) {
                        MPI_Send(MB[j], N,
MPI_INTEGER, rank + 16, 0, MPI_COMM_WORLD);
                        MPI_Send(MC[j] + H * 16, 16
* H, MPI_INTEGER, rank + 16, 0,
MPI_COMM_WORLD);
                }
                // send to (rank+8)
                for (int j = 0; j < N; j++) {
                        MPI_Send(MB[j], N,
MPI_INTEGER, rank + 8, 0, MPI_COMM_WORLD);
                        MPI_Send(MC[j] + 8 * H, 8 *
H, MPI_INTEGER, rank + 8, 0, MPI_COMM_WORLD);
                }
                // send to (rank+4)
                for (int j = 0; j < N; j++) {
                        MPI_Send(MB[j], N,
MPI_INTEGER, rank + 4, 0, MPI_COMM_WORLD);
                        MPI_Send(MC[j] + 4 * H, 4 *
H, MPI_INTEGER, rank + 4, 0, MPI_COMM_WORLD);
                }
                // send to (rank+1) | (rank+2) |
(rank+3)
                for (int i = 1; i < 4; i++) {
                        for (int j = 0; j < N; j++) {
                                MPI_Send(MB[j], N,
MPI_INTEGER, rank + i, 0, MPI_COMM_WORLD);
                                MPI_Send(MC[j] + i *
H, H, MPI_INTEGER, rank + i, 0,
MPI_COMM_WORLD);
                        }
                }
                // count MAh = MB * MCh
                for (int i = 0; i < N; ++i) {
                        for (int j = 0; j < H; ++j) {
                                MA[i][j] = 0;
                                for (int k = 0; k < N;
++k) {
                                        MA[i][j] +=
MB[i][k] * MC[k][j];
                                }
                        }
                }
                // receive from (rank+1) | (rank+2) |
(rank+3)
                for (int i = 1; i < 4; i++) {
                        for (int j = 0; j < N; j++) {
                                MPI_Recv(MA[j] + i *
H, H, MPI_INTEGER, rank + i, 0,
MPI_COMM_WORLD, &status);
                        }
                }
                // receive from (rank+4)
                for (int j = 0; j < N; j++) {
                        MPI_Recv(MA[j] + 4 * H, 4 *
H, MPI_INTEGER, rank + 4, 0, MPI_COMM_WORLD,
&status);
                }
                // receive from (rank+8)
                for (int j = 0; j < N; j++) {
                        MPI_Recv(MA[j] + 8 * H, 8 *
H, MPI_INTEGER, rank + 8, 0, MPI_COMM_WORLD,
&status);
                }
                // receive from (rank+16)
                for (int j = 0; j < N; j++) {
                        MPI_Recv(MA[j] + 16 * H, 16
* H, MPI_INTEGER, rank + 16, 0,
MPI_COMM_WORLD, &status);
                }
                // receive from (32) | (64) | (96)
```

```
		for (int i = 1; i < 4; i++) {
			for (int j = 0; j < N; j++) {
				MPI_Recv(MA[j] + i
*32* H, 32*H, MPI_INTEGER, rank + i*32, 0,
MPI_COMM_WORLD, &status);
			}
		}
		MPI_Barrier(MPI_COMM_WORLD);

		if (N <= 20) {
			cout << "MA = ";
			for (int i = 0; i < N; ++i) {
				for (int j = 0; j < N;
++j) {
					cout <<
MA[i][j] <<"  ";
				}
				cout << endl;
			}
			cout << endl;
		}
		else {
			cout << "result is too big to
print";
			cout << "MA[0][0] = " <<
MA[0][0] << endl;
		}
	}
	else if (rank == 32 || rank == 64 || rank ==
96) {
		int** MBt = new int* [N];
		int** MC32H = new int* [N];
		int** MA32H = new int* [N];

		// Receive from 0
		for (int j = 0; j < N; j++) {
			MBt[j] = new int[N];
			MC32H[j] = new int[32 * H];
			MPI_Recv(MBt[j], N,
MPI_INTEGER, 0, 0, MPI_COMM_WORLD, &status);
			MPI_Recv(MC32H[j], 32 * H,
MPI_INTEGER, 0, 0, MPI_COMM_WORLD, &status);
		}

		// send to (rank+16)
		for (int j = 0; j < N; j++) {
			MPI_Send(MBt[j], N,
MPI_INTEGER, rank + 16, 0, MPI_COMM_WORLD);
			MPI_Send(MC32H[j] + 16 *
H, 16 * H, MPI_INTEGER, rank + 16, 0,
MPI_COMM_WORLD);
		}

		// send to (rank+8)
		for (int j = 0; j < N; j++) {
			MPI_Send(MBt[j], N,
MPI_INTEGER, rank + 8, 0, MPI_COMM_WORLD);
			MPI_Send(MC32H[j] + 8 * H,
8 * H, MPI_INTEGER, rank + 8, 0,
MPI_COMM_WORLD);
		}

		// send to (rank+4)
		for (int j = 0; j < N; j++) {
			MPI_Send(MBt[j], N,
MPI_INTEGER, rank + 4, 0, MPI_COMM_WORLD);
			MPI_Send(MC32H[j] + 4 * H,
4 * H, MPI_INTEGER, rank + 4, 0,
MPI_COMM_WORLD);
		}
		// send to (rank+1) | (rank+2) |
(rank+3)
```

```
                for (int i = 1; i < 4; i++) {
                        for (int j = 0; j < N; j++) {
                                MPI_Send(MBt[j], N,
MPI_INTEGER, rank + i, 0, MPI_COMM_WORLD);
                                MPI_Send(MC32H[j]
+ i * H, H, MPI_INTEGER, rank + i, 0,
MPI_COMM_WORLD);
                        }
                }
                // count MAh = MB * MCh
                for (int i = 0; i < N; ++i) {
                        MA32H[i] = new int[N];
                        for (int j = 0; j < H; ++j) {
                                MA32H[i][j] = 0;
                                for (int k = 0; k < N;
++k) {
                                        MA32H[i][j]
+= MBt[i][k] * MC32H[k][j];
                                }
                        }
                }
                // receive from (rank+1) | (rank+2) |
(rank+3)
                for (int i = 1; i < 4; i++) {
                        for (int j = 0; j < N; j++) {
                                MPI_Recv(MA32H[j]
+ i * H, H, MPI_INTEGER, rank + i, 0,
MPI_COMM_WORLD, &status);
                        }
                }
                // receive from (rank+4)
                for (int j = 0; j < N; j++) {
                        MPI_Recv(MA32H[j] + 4 * H,
4 * H, MPI_INTEGER, rank + 4, 0,
MPI_COMM_WORLD, &status);
                }
                // receive from (rank+8)
                for (int j = 0; j < N; j++) {
                        MPI_Recv(MA32H[j] + 8 * H,
8 * H, MPI_INTEGER, rank + 8, 0,
MPI_COMM_WORLD, &status);
                }
                // receive from (rank+16)
                for (int j = 0; j < N; j++) {
                        MPI_Recv(MA32H[j] + 16 *
H, 16 * H, MPI_INTEGER, rank + 16, 0,
MPI_COMM_WORLD, &status);
                }
                //  send to (rank 0)
                for (int j = 0; j < N; j++) {
                        MPI_Send(MA32H[j], 32 * H,
MPI_INTEGER, 0, 0, MPI_COMM_WORLD);
                }
        }
        else if (rank == 16 || rank == 48 || rank ==
80 || rank == 112) {
                int** MBt = new int* [N];
                int** MC16H = new int* [N];
                int** MA16H = new int* [N];

                //Receive from (rank-16)
                for (int j = 0; j < N; j++) {
                        MBt[j] = new int[N];
                        MC16H[j] = new int[16 * H];
                        MPI_Recv(MBt[j], N,
MPI_INTEGER, rank - 16, 0, MPI_COMM_WORLD,
&status);
                        MPI_Recv(MC16H[j], 16 * H,
MPI_INTEGER, rank - 16, 0, MPI_COMM_WORLD,
&status);
                }
                // send to (rank+8)
```

```
                for (int j = 0; j < N; j++) {
                        MPI_Send(MBt[j], N,
MPI_INTEGER, rank + 8, 0, MPI_COMM_WORLD);
                        MPI_Send(MC16H[j] + 8 * H,
8 * H, MPI_INTEGER, rank + 8, 0,
MPI_COMM_WORLD);
                }
                // send to (rank+4)
                for (int j = 0; j < N; j++) {
                        MPI_Send(MBt[j], N,
MPI_INTEGER, rank + 4, 0, MPI_COMM_WORLD);
                        MPI_Send(MC16H[j] + 4 * H,
4 * H, MPI_INTEGER, rank + 4, 0,
MPI_COMM_WORLD);
                }
                // send to (rank+1) | (rank+2) |
(rank+3)
                for (int i = 1; i < 4; i++) {
                        for (int j = 0; j < N; j++) {
                                MPI_Send(MBt[j], N,
MPI_INTEGER, rank + i, 0, MPI_COMM_WORLD);
                                MPI_Send(MC16H[j]
+ i * H, H, MPI_INTEGER, rank + i, 0,
MPI_COMM_WORLD);
                        }
                }
                // count MAh = MB * MCh
                for (int i = 0; i < N; ++i) {
                        MA16H[i] = new int[N];
                        for (int j = 0; j < H; ++j) {
                                MA16H[i][j] = 0;
                                for (int k = 0; k < N;
++k) {
                                        MA16H[i][j]
+= MBt[i][k] * MC16H[k][j];
                                }
                        }
                }
                // receive from (rank+1) | (rank+2) |
(rank+3)
                for (int i = 1; i < 4; i++) {
                        for (int j = 0; j < N; j++) {
                                MPI_Recv(MA16H[j]
+ i * H, H, MPI_INTEGER, rank + i, 0,
MPI_COMM_WORLD, &status);
                        }
                }
                // receive from (rank+4)
                for (int j = 0; j < N; j++) {
                        MPI_Recv(MA16H[j] + 4 * H,
4 * H, MPI_INTEGER, rank + 4, 0,
MPI_COMM_WORLD, &status);
                }
                // receive from (rank+8)
                for (int j = 0; j < N; j++) {
                        MPI_Recv(MA16H[j] + 8 * H,
8 * H, MPI_INTEGER, rank + 8, 0,
MPI_COMM_WORLD, &status);
                }
                //  send to (rank-16)
                for (int j = 0; j < N; j++) {
                        MPI_Send(MA16H[j], 16 * H,
MPI_INTEGER, rank - 16, 0, MPI_COMM_WORLD);
                }
        }
        else if (rank == 8 || rank == 24 || rank == 40
|| rank == 56 || rank == 72 || rank == 88 || rank ==
104 || rank == 120) {
                int** MBt = new int* [N];
                int** MC8H = new int* [N];
                int** MA8H = new int* [N];

                //Receive from (rank-8)
```

```
                for (int j = 0; j < N; j++) {
                        MBt[j] = new int[N];
                        MC8H[j] = new int[8 * H];
                        MPI_Recv(MBt[j], N,
MPI_INTEGER, rank - 8, 0, MPI_COMM_WORLD,
&status);
                        MPI_Recv(MC8H[j], 8 * H,
MPI_INTEGER, rank - 8, 0, MPI_COMM_WORLD,
&status);
                }
                // send to (rank+4)
                for (int j = 0; j < N; j++) {
                        MPI_Send(MBt[j], N,
MPI_INTEGER, rank + 4, 0, MPI_COMM_WORLD);
                        MPI_Send(MC8H[j] + 4 * H, 4
* H, MPI_INTEGER, rank + 4, 0,
MPI_COMM_WORLD);
                }
                // send to (rank+1) | (rank+2) |
(rank+3)
                for (int i = 1; i < 4; i++) {
                        for (int j = 0; j < N; j++) {
                                MPI_Send(MBt[j], N,
MPI_INTEGER, rank + i, 0, MPI_COMM_WORLD);
                                MPI_Send(MC8H[j] +
i * H, H, MPI_INTEGER, rank + i, 0,
MPI_COMM_WORLD);
                        }
                }
                // count MAh = MB * MCh
                for (int i = 0; i < N; ++i) {
                        MA8H[i] = new int[N];
                        for (int j = 0; j < H; ++j) {
                                MA8H[i][j] = 0;
                                for (int k = 0; k < N;
++k) {
                                        MA8H[i][j] +=
MBt[i][k] * MC8H[k][j];
                                }
                        }
                }
                // receive from (rank+1) | (rank+2) |
(rank+3)
                for (int i = 1; i < 4; i++) {
                        for (int j = 0; j < N; j++) {
                                MPI_Recv(MA8H[j] +
i * H, H, MPI_INTEGER, rank + i, 0,
MPI_COMM_WORLD, &status);
                        }
                }
                // receive from (rank+4)
                for (int j = 0; j < N; j++) {
                        MPI_Recv(MA8H[j] + 4 * H,
4*H, MPI_INTEGER, rank + 4, 0,
MPI_COMM_WORLD, &status);
                }
                //  send to (rank-8)
                for (int j = 0; j < N; j++) {
                        MPI_Send(MA8H[j], 8 * H,
MPI_INTEGER, rank - 8, 0, MPI_COMM_WORLD);
                }
        }

        else if (rank == 4 || rank == 12 || rank == 20
|| rank == 28 || rank == 36 || rank == 44 || rank ==
52 || rank == 60 ||
                rank == 68 || rank == 76 || rank ==
84 || rank == 92 || rank == 100 || rank == 108 ||
rank == 116 || rank == 124) {
                int** MBt = new int* [N];
                int** MC4H = new int* [N];
```

```
int** MA4H = new int* [N];


//Receive from (rank-4)
for (int j = 0; j < N; j++) {
MBt[j] = new int[N];
MC4H[j] = new int[4 * H];
MPI_Recv(MBt[j], N,
MPI_INTEGER, rank - 4, 0, MPI_COMM_WORLD,
&status);
MPI_Recv(MC4H[j], 4 * H,
MPI_INTEGER, rank - 4, 0, MPI_COMM_WORLD,
&status);
}
// send to (rank+1) | (rank+2) |
(rank+3)
for (int i = 1; i < 4; i++) {
for (int j = 0; j < N; j++) {
MPI_Send(MBt[j], N,
MPI_INTEGER, rank + i, 0, MPI_COMM_WORLD);
MPI_Send(MC4H[j] +
i * H, H, MPI_INTEGER, rank + i, 0,
MPI_COMM_WORLD);
}
}
// count MAh = MB * MCh
for (int i = 0; i < N; ++i) {
MA4H[i] = new int[N];
for (int j = 0; j < H; ++j) {
MA4H[i][j] = 0;
for (int k = 0; k < N;
++k) {
MA4H[i][j] +=
MBt[i][k] * MC4H[k][j];
}
}
}
// receive from (rank+1) | (rank+2) |
(rank+3)
for (int i = 1; i < 4; i++) {
for (int j = 0; j < N; j++) {
MPI_Recv(MA4H[j]+i*H, H, MPI_INTEGER,
rank + i, 0, MPI_COMM_WORLD, &status);
}
}
// send to (rank-4)
for (int j = 0; j < N; j++) {
MPI_Send(MA4H[j], 4*H,
MPI_INTEGER, rank - 4, 0, MPI_COMM_WORLD);
}
}
else if (rank % 4 == 1) {
int** MBt = new int* [N];
int** MCH = new int* [N];
int** MAH = new int* [N];


//Receive from (rank-1)
for (int j = 0; j < N; j++) {
MBt[j] = new int[N];
MCH[j] = new int[H];
MPI_Recv(MBt[j], N,
MPI_INTEGER, rank - 1, 0, MPI_COMM_WORLD,
&status);
MPI_Recv(MCH[j], H,
MPI_INTEGER, rank - 1, 0, MPI_COMM_WORLD,
&status);
}
// count MAh = MB * MCh
for (int i = 0; i < N; ++i) {
MAH[i] = new int[N];
for (int j = 0; j < H; ++j) {
```

```
                                MAH[i][j] = 0;
                                for (int k = 0; k < N;
++k) {
                                        MAH[i][j] +=
MBt[i][k] * MCH[k][j];
                                }
                        }
                }
                // send to (rank-1)
                for (int j = 0; j < N; j++) {
                        MPI_Send(MAH[j], H,
MPI_INTEGER, rank -1, 0, MPI_COMM_WORLD);
                }
        }
        else if (rank % 4 == 2) {
                int** MBt = new int* [N];
                int** MCH = new int* [N];
                int** MAH = new int* [N];
                //Receive from (rank-2)
                for (int j = 0; j < N; j++) {
                        MBt[j] = new int[N];
                        MCH[j] = new int[H];
                        MPI_Recv(MBt[j], N,
MPI_INTEGER, rank - 2, 0, MPI_COMM_WORLD,
&status);
                        MPI_Recv(MCH[j], H,
MPI_INTEGER, rank - 2, 0, MPI_COMM_WORLD,
&status);
                }
                // count MAh = MB * MCh
                for (int i = 0; i < N; ++i) {
                        MAH[i] = new int[N];
                        for (int j = 0; j < H; ++j) {
                                MAH[i][j] = 0;
                                for (int k = 0; k < N;
++k) {
                                        MAH[i][j] +=
MBt[i][k] * MCH[k][j];
                                }
                        }
                }
                // send to (rank-2)
                for (int j = 0; j < N; j++) {
                        MPI_Send(MAH[j], H,
MPI_INTEGER, rank - 2, 0, MPI_COMM_WORLD);
                }
        }
        else if (rank % 4 == 3) {
                int** MBt = new int* [N];
                int** MCH = new int* [N];
                int** MAH = new int* [N];
                //Receive from (rank-3)
                for (int j = 0; j < N; j++) {
                        MBt[j] = new int[N];
                        MCH[j] = new int[H];
                        MPI_Recv(MBt[j], N,
MPI_INTEGER, rank - 3, 0, MPI_COMM_WORLD,
&status);
                        MPI_Recv(MCH[j], H,
MPI_INTEGER, rank - 3, 0, MPI_COMM_WORLD,
&status);
                }
                // count MAh = MB * MCh
                for (int i = 0; i < N; ++i) {
                        MAH[i] = new int[N];
                        for (int j = 0; j < H; ++j) {
                                MAH[i][j] = 0;
```

```
                                for (int k = 0; k < N;
++k) {
                                        MAH[i][j] +=
MBt[i][k] * MCH[k][j];
                                }
                        }
                }
                // send to (rank-3)
                for (int j = 0; j < N; j++) {
                        MPI_Send(MAH[j], H,
MPI_INTEGER, rank - 3, 0, MPI_COMM_WORLD);
                }
        }
        cout << "Task" << rank << " end " << endl;

        if (rank == 0) {
                std::cout << "time(" << rank << "): "
<< static_cast<double>(clock() - t_start) /
CLOCKS_PER_SEC << std::endl;
        }
        else {
                MPI_Barrier(MPI_COMM_WORLD);
        }
        if (rank == 0) {
                cout << "Press Enter...";
                string t;
                getline(cin, t);
        }
        MPI_Finalize();
        return 0;
}
void matrixFillOnes(int* matrix[N]) {
        for (int i = 0; i < N; i++) {
                matrix[i] = new int[N];
                for (int j = 0; j < N; j++) { matrix[i][j] =
1; }
        }
}
```

ДОДАТОК Є

Програмно-апаратний комплекс для матричних операцій

Лістинг задачі 2

Аркушів 11

Київ – 2020 року

```
/*
Brovchenko Anastasiya
IO-64
Diploma work. MPI.
MA = MB*(MC*MD)
10.05.2020
*/
#include "windows.h"
#include <iostream>
#include <fstream>
#include <string>
#include <mpi.h>
#include <ctime>
using namespace std;

//increase stack capathity
#pragma comment(linker, "/STACK:160000")
const int N = 3200;
const int P = 128;
const int H = N / P;

void matrixFillOnes(int* matrix[N]);

int main(int argc, char** argv)
{
	HANDLE process = GetCurrentProcess();
	DWORD_PTR processAffinityMask = 1;
	SetProcessAffinityMask(process, processAffinityMask);
	int rank, size;
	MPI_Status status;
	MPI_Init(&argc, &argv);
	MPI_Comm_size(MPI_COMM_WORLD, &size);
	MPI_Comm_rank(MPI_COMM_WORLD, &rank);
	unsigned int t_start = clock();
	cout << "Task" << rank << " start " << endl;
	if (rank == 0) {
		int** MA = new int* [N];
		int** MB = new int* [N];
		int** MC = new int* [N];
		int** MD = new int* [N];
		matrixFillOnes(MB);
		matrixFillOnes(MC);
		matrixFillOnes(MD);
		matrixFillZeros(MA);
		// send to 32, 64, 96
		for (int i = 1; i < 4; i++) {
			for (int j = 0; j < N; j++) {
				MPI_Send(MB[j], N, MPI_INTEGER, 32 * i, 0, MPI_COMM_WORLD);
				MPI_Send(MC[j], N, MPI_INTEGER, 32 * i, 0, MPI_COMM_WORLD);
				MPI_Send(MD[j] + i * 32 * H, 32 * H, MPI_INTEGER, 32 * i, 0, MPI_COMM_WORLD);
			}		}
		// send to 16
		for (int j = 0; j < N; j++) {
			MPI_Send(MB[j], N, MPI_INTEGER, rank + 16, 0, MPI_COMM_WORLD);
```

					ДП 4665.10.000 Д7		
Зм.	Арк.	№ докум.	Підпис	Дата			
Розробила		Бровченко А.В.			*Програмно-апаратний комплекс для матричних операцій* *Додаток 7* *Лістинг задачі 2*	Літ.	Аркуш 1 / Аркушів 3
Перевір.		Корочкін О.В.					
Н. контр.		Симоненко В.П.				НТУУ "КПІ", ФІОТ, каф. ОТ, гр. ІО-64	
Затверд.		Корочкін О.В.					

```
                MPI_Send(MC[j], N,
MPI_INTEGER, rank + 16, 0, MPI_COMM_WORLD);

                MPI_Send(MD[j] + H * 16, 16
* H, MPI_INTEGER, rank + 16, 0,
MPI_COMM_WORLD);

            }

            // send to (rank+8)

            for (int j = 0; j < N; j++) {

                MPI_Send(MB[j], N,
MPI_INTEGER, rank + 8, 0, MPI_COMM_WORLD);

                MPI_Send(MC[j], N,
MPI_INTEGER, rank + 8, 0, MPI_COMM_WORLD);

                MPI_Send(MD[j] + 8 * H, 8 *
H, MPI_INTEGER, rank + 8, 0, MPI_COMM_WORLD);

            }

            // send to (rank+4)

            for (int j = 0; j < N; j++) {

                MPI_Send(MB[j], N,
MPI_INTEGER, rank + 4, 0, MPI_COMM_WORLD);

                MPI_Send(MC[j], N,
MPI_INTEGER, rank + 4, 0, MPI_COMM_WORLD);

                MPI_Send(MD[j] + 4 * H, 4 *
H, MPI_INTEGER, rank + 4, 0, MPI_COMM_WORLD);

            }

            // send to (rank+1) | (rank+2) |
(rank+3)

            for (int i = 1; i < 4; i++) {

                for (int j = 0; j < N; j++) {

                    MPI_Send(MB[j], N,
MPI_INTEGER, rank + i, 0, MPI_COMM_WORLD);

                    MPI_Send(MC[j], N,
MPI_INTEGER, rank + i, 0, MPI_COMM_WORLD);

                    MPI_Send(MD[j] + i *
H, H, MPI_INTEGER, rank + i, 0,
MPI_COMM_WORLD);

                }

            }

            int** MT = new int* [N];

            // count MAh = MB * MC * MDh

            for (int i = 0; i < N; ++i) {

                MT[i] = new int[H];

                for (int j = 0; j < H; ++j) {

                    MT[i][j] = 0;

                    for (int k = 0; k < N;
++k) {

                        MT[i][j] +=
MC[i][k] * MD[k][j];

                    }

                }

            }

            for (int i = 0; i < N; ++i) {

                for (int j = 0; j < H; ++j) {

                    MA[i][j] = 0;

                    for (int k = 0; k < N;
++k) {

                        MA[i][j] +=
MB[i][k] * MT[k][j];

                    }

                }

            }

            // receive from (rank+1) | (rank+2) |
(rank+3)

            for (int i = 1; i < 4; i++) {

                for (int j = 0; j < N; j++) {

                    MPI_Recv(MA[j] + i *
H, H, MPI_INTEGER, rank + i, 0,
MPI_COMM_WORLD, &status);

                }

            }

            // receive from (rank+4)

            for (int j = 0; j < N; j++) {
```

```
                MPI_Recv(MA[j] + 4 * H, 4 *
H, MPI_INTEGER, rank + 4, 0, MPI_COMM_WORLD,
&status);
        }
        // receive from (rank+8)
        for (int j = 0; j < N; j++) {
                MPI_Recv(MA[j] + 8 * H, 8 *
H, MPI_INTEGER, rank + 8, 0, MPI_COMM_WORLD,
&status);
        }
        // receive from (rank+16)
        for (int j = 0; j < N; j++) {
                MPI_Recv(MA[j] + 16 * H, 16
* H, MPI_INTEGER, rank + 16, 0,
MPI_COMM_WORLD, &status);
        }
        // receive from (32) | (64) | (96)
        for (int i = 1; i < 4; i++) {
                for (int j = 0; j < N; j++) {
                        MPI_Recv(MA[j] + i *
32 * H, 32 * H, MPI_INTEGER, rank + i * 32, 0,
MPI_COMM_WORLD, &status);
                }
        }
        MPI_Barrier(MPI_COMM_WORLD);

        if (N <= 20) {
                cout << "MA = ";
                for (int i = 0; i < N; ++i) {
                        for (int j = 0; j < N;
++j) {
                                cout <<
MA[i][j] <<"  ";
                        }
                        cout << endl;
                }
                else {
                        cout << "result is too big to
print ";
                        cout << "MA[0][0] = " <<
MA[0][0] << endl;
                }
        }
        else if (rank == 32 || rank == 64 || rank ==
96) {
                int** MB = new int* [N];
                int** MC = new int* [N];
                int** MD = new int* [N];
                int** MA = new int* [N];

                // Receive from 0
                for (int j = 0; j < N; j++) {
                        MB[j] = new int[N];
                        MC[j] = new int[N];
                        MD[j] = new int[32 * H];
                        MPI_Recv(MB[j], N,
MPI_INTEGER, 0, 0, MPI_COMM_WORLD, &status);
                        MPI_Recv(MC[j], N,
MPI_INTEGER, 0, 0, MPI_COMM_WORLD, &status);
                        MPI_Recv(MD[j], 32 * H,
MPI_INTEGER, 0, 0, MPI_COMM_WORLD, &status);
                }
                //cout << "Task" << rank << "
received " << endl;
                // send to 16
                for (int j = 0; j < N; j++) {
                        MPI_Send(MB[j], N,
MPI_INTEGER, rank + 16, 0, MPI_COMM_WORLD);
```

```
                MPI_Send(MC[j], N,
MPI_INTEGER, rank + 16, 0, MPI_COMM_WORLD);

                MPI_Send(MD[j] + H * 16, 16
* H, MPI_INTEGER, rank + 16, 0,
MPI_COMM_WORLD);

            }

            // send to (rank+8)

            for (int j = 0; j < N; j++) {

                MPI_Send(MB[j], N,
MPI_INTEGER, rank + 8, 0, MPI_COMM_WORLD);

                MPI_Send(MC[j], N,
MPI_INTEGER, rank + 8, 0, MPI_COMM_WORLD);

                MPI_Send(MD[j] + 8 * H, 8 *
H, MPI_INTEGER, rank + 8, 0, MPI_COMM_WORLD);

            }

            // send to (rank+4)

            for (int j = 0; j < N; j++) {

                MPI_Send(MB[j], N,
MPI_INTEGER, rank + 4, 0, MPI_COMM_WORLD);

                MPI_Send(MC[j], N,
MPI_INTEGER, rank + 4, 0, MPI_COMM_WORLD);

                MPI_Send(MD[j] + 4 * H, 4 *
H, MPI_INTEGER, rank + 4, 0, MPI_COMM_WORLD);

            }

            // send to (rank+1) | (rank+2) |
(rank+3)

            for (int i = 1; i < 4; i++) {

                for (int j = 0; j < N; j++) {

                    MPI_Send(MB[j], N,
MPI_INTEGER, rank + i, 0, MPI_COMM_WORLD);

                    MPI_Send(MC[j], N,
MPI_INTEGER, rank + i, 0, MPI_COMM_WORLD);

                    MPI_Send(MD[j] + i *
H, H, MPI_INTEGER, rank + i, 0,
MPI_COMM_WORLD);

                }

            }

            int** MT = new int* [N];


            // count MAh = MB * MC * MDh
            for (int i = 0; i < N; ++i) {
                MT[i] = new int[H];
                for (int j = 0; j < H; ++j) {
                    MT[i][j] = 0;
                    for (int k = 0; k < N;
++k) {

                        MT[i][j] +=
MC[i][k] * MD[k][j];

                    }
                }
            }
            for (int i = 0; i < N; ++i) {
                MA[i] = new int[32*H];
                for (int j = 0; j < H; ++j) {
                    MA[i][j] = 0;
                    for (int k = 0; k < N;
++k) {

                        MA[i][j] +=
MB[i][k] * MT[k][j];

                    }
                }
            }
            // receive from (rank+1) | (rank+2) |
(rank+3)
            for (int i = 1; i < 4; i++) {
                for (int j = 0; j < N; j++) {
                    MPI_Recv(MA[j] + i *
H, H, MPI_INTEGER, rank + i, 0,
MPI_COMM_WORLD, &status);
                }
            }
            // receive from (rank+4)
```

```
                for (int j = 0; j < N; j++) {
                        MPI_Recv(MA[j] + 4 * H, 4 * H, MPI_INTEGER, rank + 4, 0, MPI_COMM_WORLD, &status);
                }
                // receive from (rank+8)
                for (int j = 0; j < N; j++) {
                        MPI_Recv(MA[j] + 8 * H, 8 * H, MPI_INTEGER, rank + 8, 0, MPI_COMM_WORLD, &status);
                }
                // receive from (rank+16)
                for (int j = 0; j < N; j++) {
                        MPI_Recv(MA[j] + 16 * H, 16 * H, MPI_INTEGER, rank + 16, 0, MPI_COMM_WORLD, &status);
                }
                //  send to (rank 0)
                for (int j = 0; j < N; j++) {
                        MPI_Send(MA[j], 32 * H, MPI_INTEGER, 0, 0, MPI_COMM_WORLD);
                }
        }
        else if (rank == 16 || rank == 48 || rank == 80 || rank == 112) {
                int** MB = new int* [N];
                int** MC = new int* [N];
                int** MD = new int* [N];
                int** MA = new int* [N];

                //Receive from (rank-16)
                for (int j = 0; j < N; j++) {
                        MB[j] = new int[N];
                        MC[j] = new int[N];
                        MD[j] = new int[16 * H];
                        MPI_Recv(MB[j], N, MPI_INTEGER, rank - 16, 0, MPI_COMM_WORLD, &status);
                        MPI_Recv(MC[j], N, MPI_INTEGER, rank - 16, 0, MPI_COMM_WORLD, &status);
                        MPI_Recv(MD[j], 16 * H, MPI_INTEGER, rank - 16, 0, MPI_COMM_WORLD, &status);
                }
                //cout << "Task" << rank << " received " << endl;
                // send to (rank+8)
                for (int j = 0; j < N; j++) {
                        MPI_Send(MB[j], N, MPI_INTEGER, rank + 8, 0, MPI_COMM_WORLD);
                        MPI_Send(MC[j], N, MPI_INTEGER, rank + 8, 0, MPI_COMM_WORLD);
                        MPI_Send(MD[j] + 8 * H, 8 * H, MPI_INTEGER, rank + 8, 0, MPI_COMM_WORLD);
                }
                // send to (rank+4)
                for (int j = 0; j < N; j++) {
                        MPI_Send(MB[j], N, MPI_INTEGER, rank + 4, 0, MPI_COMM_WORLD);
                        MPI_Send(MC[j], N, MPI_INTEGER, rank + 4, 0, MPI_COMM_WORLD);
                        MPI_Send(MD[j] + 4 * H, 4 * H, MPI_INTEGER, rank + 4, 0, MPI_COMM_WORLD);
                }
                // send to (rank+1) | (rank+2) | (rank+3)
                for (int i = 1; i < 4; i++) {
                        for (int j = 0; j < N; j++) {
                                MPI_Send(MB[j], N, MPI_INTEGER, rank + i, 0, MPI_COMM_WORLD);
                                MPI_Send(MC[j], N, MPI_INTEGER, rank + i, 0, MPI_COMM_WORLD);
```

```
                        MPI_Send(MD[j] + i *
H, H, MPI_INTEGER, rank + i, 0,
MPI_COMM_WORLD);
                    }
                }
                int** MT = new int* [N];

                // count MAh = MB * MC * MDh
                for (int i = 0; i < N; ++i) {
                    MT[i] = new int[H];
                    for (int j = 0; j < H; ++j) {
                        MT[i][j] = 0;
                        for (int k = 0; k < N;
++k) {
                            MT[i][j] +=
MC[i][k] * MD[k][j];
                        }
                    }
                }
                for (int i = 0; i < N; ++i) {
                    MA[i] = new int[16*H];
                    for (int j = 0; j < H; ++j) {
                        MA[i][j] = 0;
                        for (int k = 0; k < N;
++k) {
                            MA[i][j] +=
MB[i][k] * MT[k][j];
                        }
                    }
                }
                // receive from (rank+1) | (rank+2) |
(rank+3)
                for (int i = 1; i < 4; i++) {
                    for (int j = 0; j < N; j++) {
                        MPI_Recv(MA[j] + i *
H, H, MPI_INTEGER, rank + i, 0,
MPI_COMM_WORLD, &status);
                    }
                }
                // receive from (rank+4)
                for (int j = 0; j < N; j++) {
                    MPI_Recv(MA[j] + 4 * H, 4 *
H, MPI_INTEGER, rank + 4, 0, MPI_COMM_WORLD,
&status);
                }
                // receive from (rank+8)
                for (int j = 0; j < N; j++) {
                    MPI_Recv(MA[j] + 8 * H, 8 *
H, MPI_INTEGER, rank + 8, 0, MPI_COMM_WORLD,
&status);
                }
                //  send to (rank-16)
                for (int j = 0; j < N; j++) {
                    MPI_Send(MA[j], 16 * H,
MPI_INTEGER, rank - 16, 0, MPI_COMM_WORLD);
                }
            }
            else if (rank == 8 || rank == 24 || rank == 40
|| rank == 56 || rank == 72 || rank == 88 || rank ==
104 || rank == 120) {
                int** MB = new int* [N];
                int** MC = new int* [N];
                int** MD = new int* [N];
                int** MA = new int* [N];
                //Receive from (rank-8)
                for (int j = 0; j < N; j++) {
                    MB[j] = new int[N];
                    MC[j] = new int[N];
                    MD[j] = new int[8 * H];
```

```
                MPI_Recv(MB[j], N,
MPI_INTEGER, rank - 8, 0, MPI_COMM_WORLD,
&status);

                MPI_Recv(MC[j], N,
MPI_INTEGER, rank - 8, 0, MPI_COMM_WORLD,
&status);

                MPI_Recv(MD[j], 8 * H,
MPI_INTEGER, rank - 8, 0, MPI_COMM_WORLD,
&status);

        }

        //cout << "Task" << rank << "
received" << endl;

        // send to (rank+4)

        for (int j = 0; j < N; j++) {

                MPI_Send(MB[j], N,
MPI_INTEGER, rank + 4, 0, MPI_COMM_WORLD);

                MPI_Send(MC[j], N,
MPI_INTEGER, rank + 4, 0, MPI_COMM_WORLD);

                MPI_Send(MD[j] + 4 * H, 4 *
H, MPI_INTEGER, rank + 4, 0, MPI_COMM_WORLD);

        }

        // send to (rank+1) | (rank+2) |
(rank+3)

        for (int i = 1; i < 4; i++) {

                for (int j = 0; j < N; j++) {

                        MPI_Send(MB[j], N,
MPI_INTEGER, rank + i, 0, MPI_COMM_WORLD);

                        MPI_Send(MC[j], N,
MPI_INTEGER, rank + i, 0, MPI_COMM_WORLD);

                        MPI_Send(MD[j] + i *
H, H, MPI_INTEGER, rank + i, 0,
MPI_COMM_WORLD);

                }

        }

        int** MT = new int* [N];


        // count MAh = MB * MC * MDh

        for (int i = 0; i < N; ++i) {

                MT[i] = new int[H];

                for (int j = 0; j < H; ++j) {

                        MT[i][j] = 0;

                        for (int k = 0; k < N;
++k) {

                                MT[i][j] +=
MC[i][k] * MD[k][j];

                        }

                }

        }

        for (int i = 0; i < N; ++i) {

                MA[i] = new int[8*H];

                for (int j = 0; j < H; ++j) {

                        MA[i][j] = 0;

                        for (int k = 0; k < N;
++k) {

                                MA[i][j] +=
MB[i][k] * MT[k][j];

                        }

                }

        }

        // receive from (rank+1) | (rank+2) |
(rank+3)

        for (int i = 1; i < 4; i++) {

                for (int j = 0; j < N; j++) {

                        MPI_Recv(MA[j] + i *
H, H, MPI_INTEGER, rank + i, 0,
MPI_COMM_WORLD, &status);

                }

        }

        // receive from (rank+4)

        for (int j = 0; j < N; j++) {

                MPI_Recv(MA[j] + 4 * H, 4 *
H, MPI_INTEGER, rank + 4, 0, MPI_COMM_WORLD,
&status);
```

```
                }
                //  send to (rank-8)
                for (int j = 0; j < N; j++) {
                        MPI_Send(MA[j], 8 * H,
MPI_INTEGER, rank - 8, 0, MPI_COMM_WORLD);
                }
        }
        else if (rank == 4 || rank == 12 || rank == 20
|| rank == 28 || rank == 36 || rank == 44 || rank ==
52 || rank == 60 ||
                rank == 68 || rank == 76 || rank ==
84 || rank == 92 || rank == 100 || rank == 108 ||
rank == 116 || rank == 124) {
                int** MB = new int* [N];
                int** MC = new int* [N];
                int** MD = new int* [N];
                int** MA = new int* [N];
                //Receive from (rank-4)
                for (int j = 0; j < N; j++) {
                        MB[j] = new int[N];
                        MC[j] = new int[N];
                        MD[j] = new int[4 * H];
                        MPI_Recv(MB[j], N,
MPI_INTEGER, rank - 4, 0, MPI_COMM_WORLD,
&status);
                        MPI_Recv(MC[j], N,
MPI_INTEGER, rank - 4, 0, MPI_COMM_WORLD,
&status);
                        MPI_Recv(MD[j], 4 * H,
MPI_INTEGER, rank - 4, 0, MPI_COMM_WORLD,
&status);
                }
                //cout << "Task" << rank << "
received" << endl;
                // send to (rank+1) | (rank+2) |
(rank+3)
                for (int i = 1; i < 4; i++) {
                        for (int j = 0; j < N; j++) {
                                MPI_Send(MB[j], N,
MPI_INTEGER, rank + i, 0, MPI_COMM_WORLD);
                                MPI_Send(MC[j], N,
MPI_INTEGER, rank + i, 0, MPI_COMM_WORLD);
                                MPI_Send(MD[j] + i *
H, H, MPI_INTEGER, rank + i, 0,
MPI_COMM_WORLD);
                        }
                }
                int** MT = new int* [N];
                // count MAh = MB * MC * MDh
                for (int i = 0; i < N; ++i) {
                        MT[i] = new int[H];
                        for (int j = 0; j < H; ++j) {
                                MT[i][j] = 0;
                                for (int k = 0; k < N;
++k) {
                                        MT[i][j] +=
MC[i][k] * MD[k][j];
                                }
                        }
                }
                for (int i = 0; i < N; ++i) {
                        MA[i] = new int[4*H];
                        for (int j = 0; j < H; ++j) {
                                MA[i][j] = 0;
                                for (int k = 0; k < N;
++k) {
                                        MA[i][j] +=
MB[i][k] * MT[k][j];
                                }
                        }
```

```
            }
            // receive from (rank+1) | (rank+2) |
(rank+3)
            for (int i = 1; i < 4; i++) {
                for (int j = 0; j < N; j++) {
                    MPI_Recv(MA[j] + i *
H, H, MPI_INTEGER, rank + i, 0,
MPI_COMM_WORLD, &status);
                }
            }
            //  send to (rank-4)
            for (int j = 0; j < N; j++) {
                MPI_Send(MA[j], 4 * H,
MPI_INTEGER, rank - 4, 0, MPI_COMM_WORLD);
            }
        }
        else if (rank % 4 == 1) {
            int** MB = new int* [N];
            int** MC = new int* [N];
            int** MD = new int* [N];
            int** MA = new int* [N];
            //Receive from (rank-1)
            for (int j = 0; j < N; j++) {
                MB[j] = new int[N];
                MC[j] = new int[N];
                MD[j] = new int[H];
                MPI_Recv(MB[j], N,
MPI_INTEGER, rank - 1, 0, MPI_COMM_WORLD,
&status);
                MPI_Recv(MC[j], N,
MPI_INTEGER, rank - 1, 0, MPI_COMM_WORLD,
&status);
                MPI_Recv(MD[j], H,
MPI_INTEGER, rank - 1, 0, MPI_COMM_WORLD,
&status);
            }
            //cout << "Task" << rank << "
received" << endl;

            int** MT = new int* [N];

            // count MAh = MB * MC * MDh
            for (int i = 0; i < N; ++i) {
                MT[i] = new int[H];
                for (int j = 0; j < H; ++j) {
                    MT[i][j] = 0;
                    for (int k = 0; k < N;
++k) {
                        MT[i][j] +=
MC[i][k] * MD[k][j];
                    }
                }
            }
            for (int i = 0; i < N; ++i) {
                MA[i] = new int[H];
                for (int j = 0; j < H; ++j) {
                    MA[i][j] = 0;
                    for (int k = 0; k < N;
++k) {
                        MA[i][j] +=
MB[i][k] * MT[k][j];
                    }
                }
            }
            // send to (rank-1)
            for (int j = 0; j < N; j++) {
                MPI_Send(MA[j], H,
MPI_INTEGER, rank - 1, 0, MPI_COMM_WORLD);
            }
        }
```

```
        else if (rank % 4 == 2) {
                int** MB = new int* [N];
                int** MC = new int* [N];
                int** MD = new int* [N];
                int** MA = new int* [N];

                //Receive from (rank-1)
                for (int j = 0; j < N; j++) {
                        MB[j] = new int[N];
                        MC[j] = new int[N];
                        MD[j] = new int[H];
                        MPI_Recv(MB[j], N,
MPI_INTEGER, rank - 2, 0, MPI_COMM_WORLD,
&status);
                        MPI_Recv(MC[j], N,
MPI_INTEGER, rank - 2, 0, MPI_COMM_WORLD,
&status);
                        MPI_Recv(MD[j], H,
MPI_INTEGER, rank - 2, 0, MPI_COMM_WORLD,
&status);
                }
                //cout << "Task" << rank << "
received" << endl;
                int** MT = new int* [N];
                // count MAh = MB * MC * MDh
                for (int i = 0; i < N; ++i) {
                        MT[i] = new int[H];
                        for (int j = 0; j < H; ++j) {
                                MT[i][j] = 0;
                                for (int k = 0; k < N;
++k) {
                                        MT[i][j] +=
MC[i][k] * MD[k][j];
                                }
                        }
                }
                for (int i = 0; i < N; ++i) {
                        MA[i] = new int[H];
                        for (int j = 0; j < H; ++j) {
                                MA[i][j] = 0;
                                for (int k = 0; k < N;
++k) {
                                        MA[i][j] +=
MB[i][k] * MT[k][j];
                                }
                        }
                }
                // send to (rank-2)
                for (int j = 0; j < N; j++) {
                        MPI_Send(MA[j], H,
MPI_INTEGER, rank - 2, 0, MPI_COMM_WORLD);
                }
        }
        else if (rank % 4 == 3) {
                int** MB = new int* [N];
                int** MC = new int* [N];
                int** MD = new int* [N];
                int** MA = new int* [N];
                //Receive from (rank-1)
                for (int j = 0; j < N; j++) {
                        MB[j] = new int[N];
                        MC[j] = new int[N];
                        MD[j] = new int[H];
                        MPI_Recv(MB[j], N,
MPI_INTEGER, rank - 3, 0, MPI_COMM_WORLD,
&status);
                        MPI_Recv(MC[j], N,
MPI_INTEGER, rank - 3, 0, MPI_COMM_WORLD,
&status);
```

```
                MPI_Recv(MD[j], H,
MPI_INTEGER, rank - 3, 0, MPI_COMM_WORLD,
&status);
        }
        //cout << "Task" << rank << "
received " << endl;
        int** MT = new int* [N];
        // count MAh = MB * MC * MDh
        for (int i = 0; i < N; ++i) {
                MT[i] = new int[H];
                for (int j = 0; j < H; ++j) {
                        MT[i][j] = 0;
                        for (int k = 0; k < N;
++k) {
                                MT[i][j] +=
MC[i][k] * MD[k][j];
                        }
                }
        }
        for (int i = 0; i < N; ++i) {
                MA[i] = new int[H];
                for (int j = 0; j < H; ++j) {
                        MA[i][j] = 0;
                        for (int k = 0; k < N;
++k) {
                                MA[i][j] +=
MB[i][k] * MT[k][j];
                        }
                }
        }
        // send to (rank-3)
        for (int j = 0; j < N; j++) {
                MPI_Send(MA[j], H,
MPI_INTEGER, rank - 3, 0, MPI_COMM_WORLD);
        }
        cout << "Task" << rank << " end " << endl;

        if (rank == 0) {
                std::cout << "time(" << rank << "): "
<< static_cast<double>(clock() - t_start) /
CLOCKS_PER_SEC << std::endl;
        }
        else { MPI_Barrier(MPI_COMM_WORLD);}

        if (rank == 0) {
                cout << "Press Enter...";
                string t;
                getline(cin, t);
        }
        MPI_Finalize();
        return 0;
}
void matrixFillOnes(int* matrix[N]){
        for (int i = 0; i < N; i++) {
                matrix[i] = new int[N];
                for (int j = 0; j < N; j++) { matrix[i][j] =
1; }
        }
}
```

ДОДАТОК Ж

Програмно-апаратний комплекс для матричних операцій

Лістинг задачі 3

Аркушів 10

Київ – 2020 року

```
/*
Brovchenko Anastasiya
IO-64
Diploma work. MPI.
A = B*MC + D*ME
10.05.2020
*/
#include "windows.h"
#include <iostream>
#include <fstream>
#include <string>
#include <mpi.h>
#include <ctime>
using namespace std;
//increase stack capathity
#pragma comment(linker, "/STACK:1600000")
const int N = 10240;
const int P = 128;
const int H = N / P;
void matrixFillOnes(int* matrix[N]);
void vectorFillOnes(int vector[N]);
int main(int argc, char** argv)
{
	HANDLE process = GetCurrentProcess();
	DWORD_PTR processAffinityMask = 63;
	SetProcessAffinityMask(process, processAffinityMask);
	int rank, size;
	MPI_Status status;
	MPI_Init(&argc, &argv);
	MPI_Comm_size(MPI_COMM_WORLD, &size);
	MPI_Comm_rank(MPI_COMM_WORLD, &rank);
	unsigned int t_start = clock();
	cout << "Task" << rank << " start " << endl;
	if (rank == 0) {
		int* A = new int [N];
		int* B = new int [N];
		int** MC = new int* [N];
		int* D = new int[N];
		int** ME = new int* [N];
		vectorFillOnes(B);
		matrixFillOnes(MC);
		vectorFillOnes(D);
		matrixFillOnes(ME);
		// send to 32, 64, 96
		for (int i = 1; i < 4; i++) {
			MPI_Send(B, N, MPI_INTEGER, 32 * i, 0, MPI_COMM_WORLD);
			MPI_Send(D, N, MPI_INTEGER, 32 * i, 0, MPI_COMM_WORLD);
			for (int j = 0; j < N; j++) {
				MPI_Send(MC[j] + i * 32 * H, 32 * H, MPI_INTEGER, 32 * i, 0, MPI_COMM_WORLD);
				MPI_Send(ME[j] + i * 32 * H, 32 * H, MPI_INTEGER, 32 * i, 0, MPI_COMM_WORLD); } }
		// send to 16
		MPI_Send(B, N, MPI_INTEGER, 16 + rank, 0, MPI_COMM_WORLD);
```

					ДП 4665.11.000 Д8		
Зм.	Арк.	№ докум.	Підпис	Дата			
Розробила		Бровченко А.В.			Програмно-апаратний комплекс для матричних операцій Додаток 8 Лістинг задачі 3	Літ.	Аркуш 1 / Аркушів 3
Перевір.		Корочкін О.В.					
Н. контр.		Симоненко В.П.				НТУУ "КПІ", ФІОТ, каф. ОТ, гр. ІО-64	
Затверд.		Корочкін О.В.					

```
            MPI_Send(D, N, MPI_INTEGER, 16 +
rank, 0, MPI_COMM_WORLD);

            for (int j = 0; j < N; j++) {

                MPI_Send(MC[j] + H * 16, 16
* H, MPI_INTEGER, rank + 16, 0,
MPI_COMM_WORLD);

                MPI_Send(ME[j] + H * 16, 16
* H, MPI_INTEGER, rank + 16, 0,
MPI_COMM_WORLD);

            }

            // send to (rank+8)

            MPI_Send(B, N, MPI_INTEGER, 8 +
rank, 0, MPI_COMM_WORLD);

            MPI_Send(D, N, MPI_INTEGER, 8 +
rank, 0, MPI_COMM_WORLD);

            for (int j = 0; j < N; j++) {

                MPI_Send(MC[j] + 8 * H, 8 *
H, MPI_INTEGER, rank + 8, 0, MPI_COMM_WORLD);

                MPI_Send(ME[j] + 8 * H, 8 *
H, MPI_INTEGER, rank + 8, 0, MPI_COMM_WORLD);

            }

            // send to (rank+4)

            MPI_Send(B, N, MPI_INTEGER, 4 +
rank, 0, MPI_COMM_WORLD);

            MPI_Send(D, N, MPI_INTEGER, 4 +
rank, 0, MPI_COMM_WORLD);

            for (int j = 0; j < N; j++) {

                MPI_Send(MC[j] + 4 * H, 4 *
H, MPI_INTEGER, rank + 4, 0, MPI_COMM_WORLD);

                MPI_Send(ME[j] + 4 * H, 4 *
H, MPI_INTEGER, rank + 4, 0, MPI_COMM_WORLD);

            }

            // send to (rank+1) | (rank+2) |
(rank+3)

            for (int i = 1; i < 4; i++) {

                MPI_Send(B, N,
MPI_INTEGER, i + rank, 0, MPI_COMM_WORLD);

                MPI_Send(D, N,
MPI_INTEGER, i + rank, 0, MPI_COMM_WORLD);

                for (int j = 0; j < N; j++) {

                    MPI_Send(MC[j] + i *
H, H, MPI_INTEGER, rank + i, 0,
MPI_COMM_WORLD);

                    MPI_Send(ME[j] + i *
H, H, MPI_INTEGER, rank + i, 0,
MPI_COMM_WORLD);

                }

            }

            int* A1 = new int[H];

            int* A2 = new int[H];

            // count Ah = B * MCh + D * MEh

            for (int j = 0; j < H; ++j) {

                A1[j] = 0;

                A2[j] = 0;

                for (int k = 0; k < N; ++k) {

                    A1[j] += B[k] *
MC[k][j];

                    A2[j] += D[k] *
ME[k][j];

                }

                A[j] = A1[j] * A2[j];

            }

            // receive from (rank+1) | (rank+2) |
(rank+3)

            for (int i = 1; i < 4; i++) {

                MPI_Recv(A + i * H, H,
MPI_INTEGER, rank + i, 0, MPI_COMM_WORLD,
&status);

            }

            // receive from (rank+4)

            MPI_Recv(A + 4 * H, 4 * H,
MPI_INTEGER, rank + 4, 0, MPI_COMM_WORLD,
&status);

            // receive from (rank+8)
```

```
                MPI_Recv(A + 8 * H, 8 * H,
MPI_INTEGER, rank + 8, 0, MPI_COMM_WORLD,
&status);

                // receive from (rank+16)

                MPI_Recv(A + 16 * H, 16 * H,
MPI_INTEGER, rank + 16, 0, MPI_COMM_WORLD,
&status);

                // receive from (32) | (64) | (96)

                for (int i = 1; i < 4; i++) {

                        MPI_Recv(A + i*32* H, 32*H,
MPI_INTEGER, rank + i*32, 0, MPI_COMM_WORLD,
&status);

                }

                MPI_Barrier(MPI_COMM_WORLD);


                if (N <= 20) {

                        cout << "MA = ";

                        for (int i = 0; i < N; ++i) {

                                cout << A[i]
<< "  ";

                        }

                        cout << endl;

                }

                else {

                        cout << "result is too big to
print ";

                        cout << "A[0] = " << A[0] <<
endl;

                }

        }

        else if (rank == 32 || rank == 64 || rank ==
96) {

                int* A = new int[32*H];

                int* B = new int[N];

                int** MC = new int* [N];

                int* D = new int[N];

                int** ME = new int* [N];

                // Receive from 0

                MPI_Recv(B, N, MPI_INTEGER, 0, 0,
MPI_COMM_WORLD, &status);

                MPI_Recv(D, N, MPI_INTEGER, 0, 0,
MPI_COMM_WORLD, &status);

                for (int j = 0; j < N; j++) {

                        MC[j] = new int[32*H];

                        ME[j] = new int[32*H];

                        MPI_Recv(MC[j], 32 * H,
MPI_INTEGER, 0, 0, MPI_COMM_WORLD, &status);

                        MPI_Recv(ME[j], 32 * H,
MPI_INTEGER, 0, 0, MPI_COMM_WORLD, &status);

                }

                // send to (rank+16)

                MPI_Send(B, N, MPI_INTEGER, 16 +
rank, 0, MPI_COMM_WORLD);

                MPI_Send(D, N, MPI_INTEGER, 16 +
rank, 0, MPI_COMM_WORLD);

                for (int j = 0; j < N; j++) {

                        MPI_Send(MC[j] + 16 * H, 16
* H, MPI_INTEGER, rank + 16, 0,
MPI_COMM_WORLD);

                        MPI_Send(ME[j] + 16 * H, 16
* H, MPI_INTEGER, rank + 16, 0,
MPI_COMM_WORLD);

                }

                // send to (rank+8)

                MPI_Send(B, N, MPI_INTEGER, 8 +
rank, 0, MPI_COMM_WORLD);

                MPI_Send(D, N, MPI_INTEGER, 8 +
rank, 0, MPI_COMM_WORLD);

                for (int j = 0; j < N; j++) {

                        MPI_Send(MC[j] + 8 * H, 8 *
H, MPI_INTEGER, rank + 8, 0, MPI_COMM_WORLD);

                        MPI_Send(ME[j] + 8 * H, 8 *
H, MPI_INTEGER, rank + 8, 0, MPI_COMM_WORLD);
```

```
            }
            // send to (rank+4)
            MPI_Send(B, N, MPI_INTEGER, 4 +
rank, 0, MPI_COMM_WORLD);
            MPI_Send(D, N, MPI_INTEGER, 4 +
rank, 0, MPI_COMM_WORLD);
            for (int j = 0; j < N; j++) {
                MPI_Send(MC[j] + 4 * H, 4 *
H, MPI_INTEGER, rank + 4, 0, MPI_COMM_WORLD);
                MPI_Send(ME[j] + 4 * H, 4 *
H, MPI_INTEGER, rank + 4, 0, MPI_COMM_WORLD);
            }
            // send to (rank+1) | (rank+2) |
(rank+3)
            for (int i = 1; i < 4; i++) {
                MPI_Send(B, N,
MPI_INTEGER, i + rank, 0, MPI_COMM_WORLD);
                MPI_Send(D, N,
MPI_INTEGER, i + rank, 0, MPI_COMM_WORLD);
                for (int j = 0; j < N; j++) {
                    MPI_Send(MC[j] + i *
H, H, MPI_INTEGER, rank + i, 0,
MPI_COMM_WORLD);
                    MPI_Send(ME[j] + i *
H, H, MPI_INTEGER, rank + i, 0,
MPI_COMM_WORLD);
                }
            }
            int* A1 = new int[H];
            int* A2 = new int[H];
            // count Ah = B * MCh + D * MEh
            for (int j = 0; j < H; ++j) {
                A1[j] = 0;
                A2[j] = 0;
                for (int k = 0; k < N; ++k) {
                    A1[j] += B[k] *
MC[k][j];
                    A2[j] += D[k] *
ME[k][j];
                }
                A[j] = A1[j] * A2[j];
            }
            // receive from (rank+1) | (rank+2) |
(rank+3)
            for (int i = 1; i < 4; i++) {
                MPI_Recv(A + i * H, H,
MPI_INTEGER, rank + i, 0, MPI_COMM_WORLD,
&status);
            }
            // receive from (rank+4)
            MPI_Recv(A + 4 * H, 4 * H,
MPI_INTEGER, rank + 4, 0, MPI_COMM_WORLD,
&status);
            // receive from (rank+8)
            MPI_Recv(A + 8 * H, 8 * H,
MPI_INTEGER, rank + 8, 0, MPI_COMM_WORLD,
&status);
            // receive from (rank+16)
            MPI_Recv(A + 16 * H, 16 * H,
MPI_INTEGER, rank + 16, 0, MPI_COMM_WORLD,
&status);
            //  send to (rank 0)
            MPI_Send(A, 32 * H, MPI_INTEGER,
0, 0, MPI_COMM_WORLD);
        }
        else if (rank == 16 || rank == 48 || rank ==
80 || rank == 112) {
            int* A = new int[16*H];
            int* B = new int[N];
            int** MC = new int* [N];
            int* D = new int[N];
            int** ME = new int* [N];
            // Receive from rank-16
```

```
MPI_Recv(B, N, MPI_INTEGER, rank-16, 0, MPI_COMM_WORLD, &status);
MPI_Recv(D, N, MPI_INTEGER, rank - 16, 0, MPI_COMM_WORLD, &status);
for (int j = 0; j < N; j++) {
MC[j] = new int[16 * H];
ME[j] = new int[16 * H];
MPI_Recv(MC[j], 16 * H, MPI_INTEGER, rank - 16, 0, MPI_COMM_WORLD, &status);
MPI_Recv(ME[j], 16 * H, MPI_INTEGER, rank - 16, 0, MPI_COMM_WORLD, &status);
}
// send to (rank+8)
MPI_Send(B, N, MPI_INTEGER, 8 + rank, 0, MPI_COMM_WORLD);
MPI_Send(D, N, MPI_INTEGER, 8 + rank, 0, MPI_COMM_WORLD);
for (int j = 0; j < N; j++) {
MPI_Send(MC[j] + 8 * H, 8 * H, MPI_INTEGER, rank + 8, 0, MPI_COMM_WORLD);
MPI_Send(ME[j] + 8 * H, 8 * H, MPI_INTEGER, rank + 8, 0, MPI_COMM_WORLD);
}
// send to (rank+4)
MPI_Send(B, N, MPI_INTEGER, 4 + rank, 0, MPI_COMM_WORLD);
MPI_Send(D, N, MPI_INTEGER, 4 + rank, 0, MPI_COMM_WORLD);
for (int j = 0; j < N; j++) {
MPI_Send(MC[j] + 4 * H, 4 * H, MPI_INTEGER, rank + 4, 0, MPI_COMM_WORLD);
MPI_Send(ME[j] + 4 * H, 4 * H, MPI_INTEGER, rank + 4, 0, MPI_COMM_WORLD);
}
// send to (rank+1) | (rank+2) | (rank+3)
for (int i = 1; i < 4; i++) {
MPI_Send(B, N, MPI_INTEGER, i + rank, 0, MPI_COMM_WORLD);
MPI_Send(D, N, MPI_INTEGER, i + rank, 0, MPI_COMM_WORLD);
for (int j = 0; j < N; j++) {
MPI_Send(MC[j] + i * H, H, MPI_INTEGER, rank + i, 0, MPI_COMM_WORLD);
MPI_Send(ME[j] + i * H, H, MPI_INTEGER, rank + i, 0, MPI_COMM_WORLD);
}
}
int* A1 = new int[H];
int* A2 = new int[H];
// count Ah = B * MCh + D * MEh
for (int j = 0; j < H; ++j) {
A1[j] = 0;
A2[j] = 0;
for (int k = 0; k < N; ++k) {
A1[j] += B[k] * MC[k][j];
A2[j] += D[k] * ME[k][j];
}
A[j] = A1[j] * A2[j];
}
// receive from (rank+1) | (rank+2) | (rank+3)
for (int i = 1; i < 4; i++) {
MPI_Recv(A + i * H, H, MPI_INTEGER, rank + i, 0, MPI_COMM_WORLD, &status);
}
// receive from (rank+4)
```

```
            MPI_Recv(A + 4 * H, 4 * H,
MPI_INTEGER, rank + 4, 0, MPI_COMM_WORLD,
&status);

            // receive from (rank+8)

            MPI_Recv(A + 8 * H, 8 * H,
MPI_INTEGER, rank + 8, 0, MPI_COMM_WORLD,
&status);

            //  send to (rank-16)

            MPI_Send(A, 16 * H, MPI_INTEGER,
rank - 16, 0, MPI_COMM_WORLD);

        }

        else if (rank == 8 || rank == 24 || rank == 40
|| rank == 56 || rank == 72 || rank == 88 || rank ==
104 || rank == 120) {

            int* A = new int[8*N];

            int* B = new int[N];

            int** MC = new int* [N];

            int* D = new int[N];

            int** ME = new int* [N];

            // Receive from rank-8

            MPI_Recv(B, N, MPI_INTEGER, rank -
8, 0, MPI_COMM_WORLD, &status);

            MPI_Recv(D, N, MPI_INTEGER, rank -
8, 0, MPI_COMM_WORLD, &status);

            for (int j = 0; j < N; j++) {

                MC[j] = new int[8 * H];

                ME[j] = new int[8 * H];

                MPI_Recv(MC[j], 8 * H,
MPI_INTEGER, rank - 8, 0, MPI_COMM_WORLD,
&status);

                MPI_Recv(ME[j], 8 * H,
MPI_INTEGER, rank - 8, 0, MPI_COMM_WORLD,
&status);

            }

            // send to (rank+4)

            MPI_Send(B, N, MPI_INTEGER, 4 +
rank, 0, MPI_COMM_WORLD);

            MPI_Send(D, N, MPI_INTEGER, 4 +
rank, 0, MPI_COMM_WORLD);

            for (int j = 0; j < N; j++) {

                MPI_Send(MC[j] + 4 * H, 4 *
H, MPI_INTEGER, rank + 4, 0, MPI_COMM_WORLD);

                MPI_Send(ME[j] + 4 * H, 4 *
H, MPI_INTEGER, rank + 4, 0, MPI_COMM_WORLD);

            }

            // send to (rank+1) | (rank+2) |
(rank+3)

            for (int i = 1; i < 4; i++) {

                MPI_Send(B, N,
MPI_INTEGER, i + rank, 0, MPI_COMM_WORLD);

                MPI_Send(D, N,
MPI_INTEGER, i + rank, 0, MPI_COMM_WORLD);

                for (int j = 0; j < N; j++) {

                    MPI_Send(MC[j] + i *
H, H, MPI_INTEGER, rank + i, 0,
MPI_COMM_WORLD);

                    MPI_Send(ME[j] + i *
H, H, MPI_INTEGER, rank + i, 0,
MPI_COMM_WORLD);

                }

            }

            int* A1 = new int[H];

            int* A2 = new int[H];

            // count Ah = B * MCh + D * MEh

            for (int j = 0; j < H; ++j) {

                A1[j] = 0;

                A2[j] = 0;

                for (int k = 0; k < N; ++k) {

                    A1[j] += B[k] *
MC[k][j];

                    A2[j] += D[k] *
ME[k][j];

                }

                A[j] = A1[j] * A2[j];
```

```
		}

		// receive from (rank+1) | (rank+2) | (rank+3)

		for (int i = 1; i < 4; i++) {

			MPI_Recv(A + i * H, H, MPI_INTEGER, rank + i, 0, MPI_COMM_WORLD, &status);

		}

		// receive from (rank+4)

		MPI_Recv(A + 4 * H, 4 * H, MPI_INTEGER, rank + 4, 0, MPI_COMM_WORLD, &status);

		//  send to (rank-8)

		MPI_Send(A, 8 * H, MPI_INTEGER, rank - 8, 0, MPI_COMM_WORLD);

	}

	else if (rank == 4 || rank == 12 || rank == 20 || rank == 28 || rank == 36 || rank == 44 || rank == 52 || rank == 60 ||

		rank == 68 || rank == 76 || rank == 84 || rank == 92 || rank == 100 || rank == 108 || rank == 116 || rank == 124) {

		int* A = new int[4*H];

		int* B = new int[N];

		int** MC = new int* [N];

		int* D = new int[N];

		int** ME = new int* [N];

		// Receive from rank-4

		MPI_Recv(B, N, MPI_INTEGER, rank - 4, 0, MPI_COMM_WORLD, &status);

		MPI_Recv(D, N, MPI_INTEGER, rank - 4, 0, MPI_COMM_WORLD, &status);

		for (int j = 0; j < N; j++) {

			MC[j] = new int[4 * H];

			ME[j] = new int[4 * H];

			MPI_Recv(MC[j], 4 * H, MPI_INTEGER, rank - 4, 0, MPI_COMM_WORLD, &status);

			MPI_Recv(ME[j], 4 * H, MPI_INTEGER, rank - 4, 0, MPI_COMM_WORLD, &status);

		}

		// send to (rank+1) | (rank+2) | (rank+3)

		for (int i = 1; i < 4; i++) {

			MPI_Send(B, N, MPI_INTEGER, i + rank, 0, MPI_COMM_WORLD);

			MPI_Send(D, N, MPI_INTEGER, i + rank, 0, MPI_COMM_WORLD);

			for (int j = 0; j < N; j++) {

				MPI_Send(MC[j] + i * H, H, MPI_INTEGER, rank + i, 0, MPI_COMM_WORLD);

				MPI_Send(ME[j] + i * H, H, MPI_INTEGER, rank + i, 0, MPI_COMM_WORLD);

			}

		}

		int* A1 = new int[H];

		int* A2 = new int[H];

		// count Ah = B * MCh + D * MEh

		for (int j = 0; j < H; ++j) {

			A1[j] = 0;

			A2[j] = 0;

			for (int k = 0; k < N; ++k) {

				A1[j] += B[k] * MC[k][j];

				A2[j] += D[k] * ME[k][j];

			}

			A[j] = A1[j] * A2[j];

		}
```

```
            // receive from (rank+1) | (rank+2) |
(rank+3)
            for (int i = 1; i < 4; i++) {
                  MPI_Recv(A + i * H, H,
MPI_INTEGER, rank + i, 0, MPI_COMM_WORLD,
&status);
            }
            //  send to (rank-4)
            MPI_Send(A, 4*H, MPI_INTEGER,
rank - 4, 0, MPI_COMM_WORLD);
      }
      else if (rank % 4 == 1) {
            int* A = new int[H];
            int* B = new int[N];
            int** MC = new int* [N];
            int* D = new int[N];
            int** ME = new int* [N];
            // Receive from rank-1
            MPI_Recv(B, N, MPI_INTEGER, rank -
1, 0, MPI_COMM_WORLD, &status);
            MPI_Recv(D, N, MPI_INTEGER, rank -
1, 0, MPI_COMM_WORLD, &status);
            for (int j = 0; j < N; j++) {
                  MC[j] = new int[H];
                  ME[j] = new int[H];
                  MPI_Recv(MC[j], H,
MPI_INTEGER, rank - 1, 0, MPI_COMM_WORLD,
&status);
                  MPI_Recv(ME[j], H,
MPI_INTEGER, rank - 1, 0, MPI_COMM_WORLD,
&status);
            }
            int* A1 = new int[H];
            int* A2 = new int[H];
            // count Ah = B * MCh + D * MEh
            for (int j = 0; j < H; ++j) {
                  A1[j] = 0;
                  A2[j] = 0;
                  for (int k = 0; k < N; ++k) {
                        A1[j] += B[k] *
MC[k][j];
                        A2[j] += D[k] *
ME[k][j];
                  }
                  A[j] = A1[j] * A2[j];
            }
            // send to (rank-1)
            MPI_Send(A, H, MPI_INTEGER, rank -
1, 0, MPI_COMM_WORLD);
      }
      else if (rank % 4 == 2) {
            int* A = new int[N];
            int* B = new int[N];
            int** MC = new int* [N];
            int* D = new int[N];
            int** ME = new int* [N];
            // Receive from rank-2
            MPI_Recv(B, N, MPI_INTEGER, rank -
2, 0, MPI_COMM_WORLD, &status);
            MPI_Recv(D, N, MPI_INTEGER, rank -
2, 0, MPI_COMM_WORLD, &status);
            for (int j = 0; j < N; j++) {
                  MC[j] = new int[H];
                  ME[j] = new int[H];
                  MPI_Recv(MC[j], H,
MPI_INTEGER, rank - 2, 0, MPI_COMM_WORLD,
&status);
                  MPI_Recv(ME[j], H,
MPI_INTEGER, rank - 2, 0, MPI_COMM_WORLD,
&status);
            }
```

```
            int* A1 = new int[H];
            int* A2 = new int[H];
            // count Ah = B * MCh + D * MEh
            for (int j = 0; j < H; ++j) {
                A1[j] = 0;
                A2[j] = 0;
                for (int k = 0; k < N; ++k) {
                    A1[j] += B[k] *
MC[k][j];
                    A2[j] += D[k] *
ME[k][j];
                }
                A[j] = A1[j] * A2[j];
            }
            // send to (rank-2)
            MPI_Send(A, H, MPI_INTEGER, rank -
2, 0, MPI_COMM_WORLD);
        }
        else if (rank % 4 == 3) {
            int* A = new int[H];
            int* B = new int[N];
            int** MC = new int* [N];
            int* D = new int[N];
            int** ME = new int* [N];
            // Receive from rank-2
            MPI_Recv(B, N, MPI_INTEGER, rank -
3, 0, MPI_COMM_WORLD, &status);
            MPI_Recv(D, N, MPI_INTEGER, rank -
3, 0, MPI_COMM_WORLD, &status);
            for (int j = 0; j < N; j++) {
                MC[j] = new int[H];
                ME[j] = new int[H];
                MPI_Recv(MC[j], H,
MPI_INTEGER, rank - 3, 0, MPI_COMM_WORLD,
&status);
                MPI_Recv(ME[j], H,
MPI_INTEGER, rank - 3, 0, MPI_COMM_WORLD,
&status);
            }
            int* A1 = new int[H];
            int* A2 = new int[H];
            // count Ah = B * MCh + D * MEh
            for (int j = 0; j < H; ++j) {
                A1[j] = 0;
                A2[j] = 0;
                for (int k = 0; k < N; ++k) {
                    A1[j] += B[k] *
MC[k][j];
                    A2[j] += D[k] *
ME[k][j];
                }
                A[j] = A1[j] * A2[j];
            }
            // send to (rank-2)
            MPI_Send(A, H, MPI_INTEGER, rank -
3, 0, MPI_COMM_WORLD);
        }
        cout << "Task" << rank << " end " << endl;
        if (rank == 0) {
            std::cout << "time(" << rank << "): "
<< static_cast<double>(clock() - t_start) /
CLOCKS_PER_SEC << std::endl;
        }
        else {
            MPI_Barrier(MPI_COMM_WORLD);
        }
        if (rank == 0) {
            cout << "Press Enter...";
            string t;
            getline(cin, t);
```

```
        }
        MPI_Finalize();
        return 0;
}
void matrixFillOnes(int* matrix[N]) {
        for (int i = 0; i < N; i++) {
                matrix[i] = new int[N];
                for (int j = 0; j < N; j++) { matrix[i][j] =
1; }
        }
}
void vectorFillOnes(int vector[N]) {
        for (int i = 0; i < N; i++) { vector[i] = 1; }
}
```